

MENU

SEARCH

INDEX

DETAIL

JAPANESE

1 / 1

J1017 U.S. PTO

09/918639



## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-328134

(43)Date of publication of application : 30.11.1999

(51)Int.Cl.

G06F 15/163  
G06F 13/00

(21)Application number : 10-132364

(71)Applicant : HITACHI LTD

(22)Date of filing : 14.05.1998

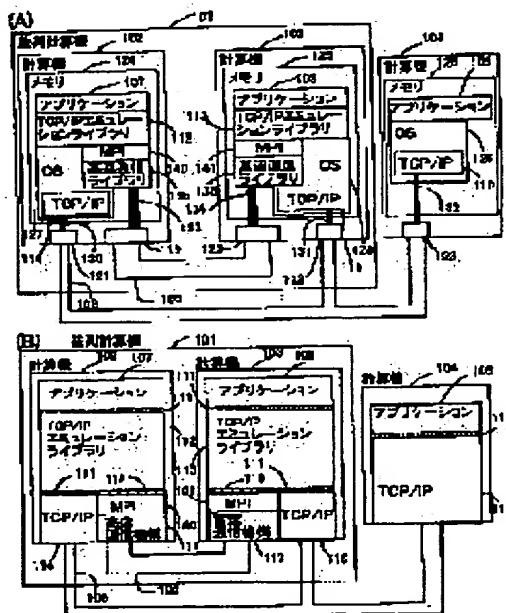
(72)Inventor : IMAKI TSUNEYUKI  
SAGAWA NOBUTOSHI

## (54) METHOD FOR TRANSMITTING AND RECEIVING DATA BETWEEN COMPUTERS

## (57)Abstract:

PROBLEM TO BE SOLVED: To utilize a message passing type stream communication from a TCP/IP application program.

SOLUTION: An emulation library 112 provided for each computer decides whether or not a communication party process that a communication request from an application program 107 specifies is inside or outside a parallel computer 101 and when the process is inside the computer, a fast communication library 135 is actuated to transmit and receive data by a message passing type communication system through a fast internal communication network 105. The data are transmitted and received so as to actualize a stream communication. When the communication party is an external computer 104 of the parallel computer 101, the library 112 requests a communication of a TCP/IP process routine 114 according to TCP/IP.



## LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

**THIS PAGE BLANK (USPTO)**

4-3  
E6095

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平11-328134

(43) 公開日 平成11年(1999)11月30日

(51) Int.Cl.<sup>6</sup>

G 0 6 F 15/163  
13/00

識別記号

3 5 1

F I

G 0 6 F 15/16  
13/00

3 1 0 R  
3 5 1 E

審査請求 未請求 請求項の数12 O L (全 23 頁)

J1017 U.S. PRO  
09/918639  
08/81/81

(21) 出願番号 特願平10-132364

(22) 出願日 平成10年(1998) 5月14日

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 今木 常之

東京都国分寺市東恋ヶ窪一丁目280番地

株式会社日立製作所中央研究所内

(72) 発明者 佐川 暢俊

東京都国分寺市東恋ヶ窪一丁目280番地

株式会社日立製作所中央研究所内

(74) 代理人 弁理士 高橋 明夫 (外1名)

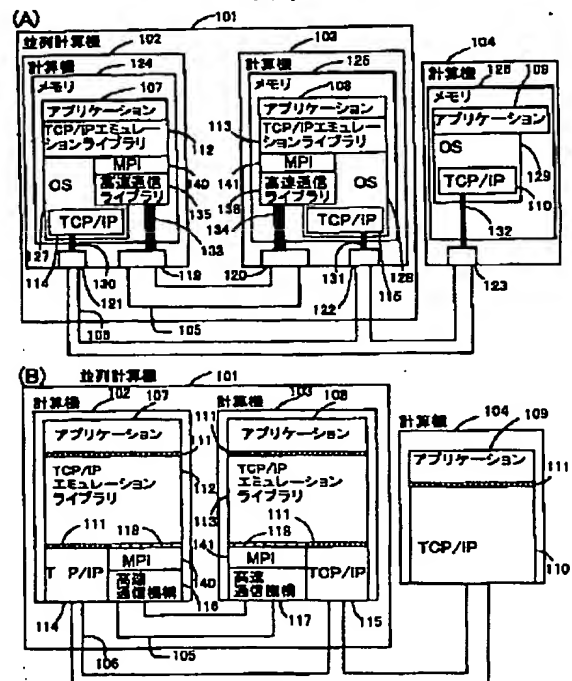
(54) 【発明の名称】 計算機間データ送受信方法

(57) 【要約】

【課題】 TCP/IPアプリケーションプログラムからメッセージパッシング型のストリーム通信を利用可能にする。

【解決手段】 各計算機に設けられたエミュレーションライブラリ112が、アプリケーションプログラム107からの通信要求が指定する通信相手プロセスが並列計算機101の内部であるか外部であるかを判別し、内部であれば高速通信ライブラリ135を起動して、高速内部通信網105を介してメッセージパッシング型通信方式でもってデータを送受信する。このデータの送受信は、ストリーム通信を実現するように実行される。相手先が並列計算機101の外部計算機104であるときには、ライブラリ112は、TCP/IP処理ルーティン114にTCP/IPに従い通信を要求する。

図1



**【特許請求の範囲】**

【請求項 1】第 1 の計算機上で実行されている第 1 のアプリケーションプログラムが発行した複数の送信命令により指定される複数の送信データのそれぞれを、第 2 の計算機上で実行されている第 2 のアプリケーションプログラムが発行した複数の受信命令に回答して、第 2 の計算機上に設けられたエミュレーションライブラリの制御下でメッセージパッシング型の通信で受信し、上記複数の送信データの連なりからなるひと繋りのデータの内、上記複数の受信命令により指定されるサイズ部分に区分して得られる部分をそれぞれの受信命令が指定する複数のバッファに格納するように、受信されたそれぞれの送信データを上記エミュレーションライブラリの制御下で処理する計算機間データ送受信方法。

【請求項 2】メッセージパッシング型の通信を実行するための第 1 の通信ライブラリを有する第 1 の計算機上で実行されている第 1 のアプリケーションプログラムが発行した複数の送信命令に回答して、それらの送信命令により指定される複数のバッファに保持された複数の送信データの送信をそれぞれ要求する複数の送信命令を、上記第 1 の計算機に設けられた第 1 のエミュレーションライブラリにより上記第 1 の通信ライブラリに対して発行し、

メッセージパッシング型の通信を実行するための第 2 の通信ライブラリを有する第 2 の計算機上で実行されている第 2 のアプリケーションプログラムが発行した複数の受信命令に回答して、上記複数の送信データをメッセージパッシング型の通信で受信するための複数の受信命令を、上記第 2 の計算機に設けられた第 2 のエミュレーションライブラリにより上記第 2 の通信ライブラリに対して発行し、

第 1 のエミュレーションライブラリが発行した上記複数の送信命令により指定される複数の送信データの連なりからなるひと繋りのデータを、上記複数の受信命令により指定されるサイズ部分に区分してそれぞれの受信命令が指定する複数のバッファに格納するように、第 2 のエミュレーションライブラリが発行した上記複数の受信命令により受信される上記複数の送信データの第 2 の計算機内での格納位置およびそれぞれの送信データの受信後の第 2 の計算機内での移動を、上記第 2 のエミュレーションライブラリにより制御する計算機間データ送受信方法。

【請求項 3】メッセージパッシング型の通信を実行するための第 1 の通信ライブラリを有する第 1 の計算機上で実行されている第 1 のアプリケーションプログラムが発行した複数回の送信命令に回答して、それらの送信命令により指定される複数のバッファに保持された複数の送信データの送信をそれぞれ要求する複数の送信命令を、上記第 1 の計算機に設けられた第 1 のエミュレーションライブラリにより上記第 1 の通信ライブラリに対して発

行し、

メッセージパッシング型の通信を実行するための第 2 の通信ライブラリを有する第 2 の計算機上で実行されている第 2 のアプリケーションプログラムが発行した複数の受信命令に回答して、上記複数の送信データをメッセージパッシング型の通信で受信するための複数の受信命令を、上記第 2 の計算機に設けられた第 2 のエミュレーションライブラリにより上記第 2 の通信ライブラリに対して発行し、

第 1 のエミュレーションライブラリが発行した上記複数の送信命令が指定する複数の送信データを連ねて構成されるひと繋りのデータを、受信側のアプリケーションプログラムが発行した複数回の受信命令により指定される複数のバッファに、それぞれの受信命令が指定するサイズ部分に区分して格納できるように、上記第 2 のエミュレーションライブラリにより (a) 第 2 のエミュレーションライブラリが発行した上記複数の受信命令の各々が指定する、受信データを格納するバッファのアドレスを、現在処理中の第 2 のアプリケーションプログラムが発行した受信命令が指定するバッファに属するアドレスもしくは上記第 2 の通信ライブラリが利用可能なバッファに属するアドレスとなるように定め、かつ (b) 、上記第 2 の通信ライブラリが利用可能な上記バッファのアドレスを指定する、第 2 のエミュレーションライブラリが発行した受信命令により受信した送信データのその後の上記第 1 の計算機内での移動を制御し、

第 2 のエミュレーションライブラリが発行した上記複数の受信命令は、それぞれ第 1 のエミュレーションライブラリが発行した上記複数の送信命令の一つに対応して発行され、

第 2 のエミュレーションライブラリが発行した上記複数の受信命令の各々が指定する受信データのサイズは、第 1 のエミュレーションライブラリが発行した上記複数の送信命令の内の対応する一つの送信命令が指定する送信データのサイズに等しくなるように設定される計算機間データ送受信方法。

【請求項 4】(a) 送信側のアプリケーションが指定するいずれかの送信データの送信を要求する、上記メッセージパッシング型の通信により定められた送信命令を送信側の計算機内で発行し、(b) 上記送信命令が指定する送信データ長を、その送信データの送信前に、受信側の計算機上で動作する受信側のエミュレーションライブラリにより検知し、(c) 受信側のアプリケーションが発行した受信命令によって指定された受信データ長と上記送信データ長とを、上記受信側のエミュレーションライブラリにより比較し、(d) 上記比較の結果に応じて、上記受信側の計算機内のエミュレーション用のバッファに上記送信データを受信することを要求するもしくは上記送信データを上記アプリケーションバッファに受信することを要求する、上記メッセージパッシング型

の通信により定められた受信命令を上記受信側のエミュレーションライブラリにより発行し、(e)上記送信命令と受信側のエミュレーションライブラリにより発行した上記受信命令とに応答して、上記送信データを上記送信側の計算機と上記受信側の計算機により、上記送信側の計算機から上記エミュレーション用のバッファもしくは上記アプリケーションバッファ内に転送する計算機間データ送受信方法。

【請求項5】(f)上記送信データが上記エミュレーションバッファに送信されたときには、そのエミュレーションバッファから上記受信データ長分だけのデータを上記受信命令が指定したアプリケーションバッファ内に、上記受信側のエミュレーションライブラリによりコピーするステップをさらに有する請求項4に記載の計算機間データ送受信方法。

【請求項6】(g)上記ステップ(e)により、上記送信データが上記アプリケーションバッファ内に転送された場合、上記送信側の計算機が後続の送信データに関してステップ(a)を実行するのを待ち、(h)上記後続の送信データに関して、ステップ(a)が実行された場合、その後続の送信データと上記受信命令が要求した受信データ長の残りの長さのデータに関して上記ステップ(b)以降のステップを実行するステップをさらに有する請求項5に記載の計算機間データ送受信方法。

【請求項7】(i)上記ステップ(a)を実行する前に、上記エミュレーションバッファ内に受信済みでいずれのアプリケーションバッファにも転送されていないデータが残っているか否かを判別し、(j)そのデータが残っているときには、そのバッファから上記アプリケーションに上記受信データ長を超えない範囲でコピーするステップをさらに有する請求項6に記載の計算機間データ送受信方法。

【請求項8】第1の通信網で相互に結合されている複数の計算機を有し、少なくともその一部の計算機が第2の通信網に相互に接続されている計算機システムにおいて、(a)上記一部の計算機の内第1の計算機上で動作する第1のアプリケーションプログラムから、第2の計算機上で動作するいずれかの第2のアプリケーションプログラムとの間のデータ送受信に使用する通信路を、第1の通信網を使用するために定められた通信規約に従って動作する処理ルーチンにより決定し、(b)第1のアプリケーションプログラムからの、上記決定された通信路に対する接続要求をエミュレーションライブラリにより処理し、その処理の中で、上記第2のアプリケーションプログラムとの通信に上記第2の通信網を利用可能か否かを判別し、(c)上記第2の通信網が利用可能と判別されたときには、上記通信路を上記エミュレーションライブラリ内に登録し、(d)その後上記第1のアプリケーションプログラムがデータ送受信命令を発行したときに、その送受信命令により指定された通信路が、上

記エミュレーションライブラリ内に登録されているか否かを判別し、(e)その指定された通信路が、上記エミュレーションライブラリ内に登録されているときには、上記エミュレーションライブラリにより、上記送受信命令が要求する送受信処理を上記第2の通信網を使用して実行し、(f)その指定された通信路が、上記エミュレーションライブラリ内に登録されていないときには、上記エミュレーションライブラリにより、上記送受信命令が要求する送受信処理の実行を上記処理ルーチンに要求し、上記処理ルーチンにより、その送受信処理を上記第1の通信網を使用して実行するデータ送受信方法。

【請求項9】上記通信規約はTCP/IPであり、上記通信路の決定は、ソケットの生成およびソケットへの名前付けの処理を含む請求項8記載のデータ送受信方法。

【請求項10】上記第2の通信網を使用した通信はメッセージパッシング型の通信である請求項8記載のデータ送受信方法。

【請求項11】第1の通信網で相互に結合されている複数の計算機を有し、少なくともその一部の計算機が第1の通信網よりも高速にデータを転送可能な第2の通信網に相互に接続され、上記一部の計算機の内第1の計算機上で動作する第1のアプリケーションプログラムから、第2の計算機上で動作するいずれかの第2のアプリケーションプログラムとの間のデータ送受信に、上記第2の通信網を使用可能であるときには、これを使用し、そうでないときには上記第2の通信網を使用する計算機システムにおいて、

第1のアプリケーションプログラムに宛てて他のアプリケーションプログラムから送信されるデータの有無を検出するときに、上記第2の通信網を介して送信されるデータの検出処理と、上記第1の通信網を介して送信されるデータの検出処理とを、ノンブロックで交互に繰り返す、

両検出処理を1回繰り返す毎に両検出処理を中断し、上記第1の計算機上で動作している他のプロセスに処理を譲渡する送信データ検出方法。

【請求項12】第1の通信網で相互に結合されている複数の計算機を有し、少なくともその一部の計算機が第1の通信網よりも高速にデータを転送可能な第2の通信網に相互に接続され、上記一部の計算機の内第1の計算機上で動作する第1のアプリケーションプログラムから、第2の計算機上で動作するいずれかの第2のアプリケーションプログラムとの間のデータ送受信に、上記第2の通信網を使用可能であるときには、これを使用し、そうでないときには上記第2の通信網を使用する計算機システムにおいて、

第1のアプリケーションプログラムに宛てて他のアプリケーションプログラムから送信されるデータの有無を検出するときに、二つのスレッドを生成し、

上記第2の通信網を介して送信されるデータの検出処理と、上記第1の通信網を介して送信されるデータの検出処理とを、それぞれのスレッド上で独立に行なうデータ検出方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、複数の種類の通信網により接続された複数の計算機を有する計算機システムにおける、計算機間のデータ送受信方法に係り、特にメッセージパッシング型の通信を実行するのに好適な、計算機間のデータ送受信方法に関する。

【0002】

【従来の技術】計算機間の通信規約としてTCP/IPが極めて一般的に使用されている。TCP/IPを使用して他のプログラムと通信するように構成されたプログラムを以下ではTCP/IPアプリケーションプログラムと呼ぶ。このTCP/IPアプリケーションプログラムからTCP/IP以外の通信規約を使用できるようにした計算機システムも存在する。すなわち、計算機システムを構成する計算機が性能の異なる複数のネットワークにより構成され、そのシステム内の計算機が他の計算機と通信するときに、それらの計算機がその通信にこれらのネットワークのいずれを使用するかに依存して、TCP/IPまたは他の通信規約を使用するシステムも提案されている。例えば、バークレー大のSteven H. Rodriguesらが開発したシステムは、TCP/IPプロトコルで使用可能な広域のネットワークと、それより狭い領域でより簡潔でオーバーヘッドが低い通信規約を使用する局所的なネットワークからなり、局所的なネットワークに接続された計算機同士は、この通信規約を使用して通信し、広域のネットワークに接続された計算機同士は、TCP/IP規約に従って通信する。たとえば、"High-Performance Local-Area Communication With Fast Socket", USENIX '97 Annual Technical Conference pp. 257-274) 参照。

【0003】この計算機システムには、TCP/IPアプリケーションプログラムからもこの通信規約を利用できるようにするために、TCP/IPエミュレーションライブラリが用意されている。上記システムではこのTCP/IPエミュレーションライブラリは、高速ソケット(Fast Socket)と呼ばれている。

【0004】高速ソケットは、簡潔な通信規約としてワークステーションクラスタ向けに開発されたアクティブメッセージ(Active Message)を用いる。例えば 文献 T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser "Active Messages: a Mechanism for Integrated Communication and Computation", in Proceedings of the 19th International Symposium on Computer Architecture, May 1992 pp. 256-266 参照。アクティブメッセージでは、データを送る側のア

プリケーションプログラムが受け手側のアプリケーションプログラムに割り込み、受け手側がその割り込みを契機にデータ受信処理を行うという方法でデータをやり取りする。

【0005】並列計算機は、複数の計算機が互いに通信しながら協調して1つの問題を解決することを目的とした計算機である。この目的を満たすため、一般に並列計算機の内部の各計算機は高速の内部高速通信網により相互に接続され、並列計算機内の少なくとも一部の計算機は、さらに、LAN等のより広域のネットワークにより外部の計算機に接続されている。

【0006】広域のネットワークに使用される通信規約は主としてTCP/IPである。並列計算機は、内部高速通信網を使用するために、各計算機に他の内部計算機と内部高速通信網を介して通信するための高速な通信ハードウェアとそのハードウェアを使用するための高速通信ライブラリを設けている。現在、多くの並列計算機に採用されている通信規約は、メッセージパッシング型の通信である。メッセージパッシング型の通信は、送信アプリケーションプログラムが発行する送信命令と受信アプリケーションプログラムが発行する受信命令が一对一に対応付けられた時に通信が行われるという通信である。多くの場合、この通信方式は並列計算機内部における高速通信ハードウェアに適している。このメッセージパッシング型通信を実現するために使用される高速通信ライブラリには、リモートメモリ書き込みライブラリあるいはPUTライブラリと呼ばれているものが主に使用されている。

【0007】並列計算機内部の通信のオーバーヘッドは、高速通信ハードウェアを使用することによりかなり減じることができる。アクティブメッセージで用いる割り込み型の通信方式では、割り込みのオーバーヘッドが目立ってしまう。しかるに、メッセージパッシング型通信方式は、割り込みを前提としないので、並列計算機の内部での通信にはアクティブメッセージ型通信よりも適している。

【0008】

【発明が解決しようとする課題】前述のように、並列計算機の内部高速通信網を使用するための通信規約は一般にはメッセージパッシング型である。しかし、並列計算機をビジネスの分野で利用する場合に使用するビジネス用のアプリケーションプログラムは多くはTCP/IP規約を使用するように構成されている。したがって、そのようなアプリケーションプログラムは、そのままではメッセージパッシング型通信方式をそのまま利用することはできない。しかも、メッセージパッシング型通信方式を利用した状態で、ストリーム通信を実現する方法が知られていない。また、割込を使用するアクティブメッセージ型通信で使用されるストリーム通信の実現方法をそのままメッセージパッシング型通信でのストリーム通

信に利用するわけには行かない。

【0009】従って本発明の目的は、メッセージパッシング型通信を実行するように構成されている計算機上で動作する複数のアプリケーションプログラム間でストリーム通信を実行可能にする計算機間データ送受信方法を提供することである。

【0010】本発明のより具体的な目的は、第1の計算機ネットワークを介して、TCP/IPのごとき、メッセージパッシング型通信と異なる通信規約を用いて通信を実行するように構成されている複数のアプリケーションプログラム間で、その計算機ネットワークよりも高速な通信網を使用して、かつ、メッセージパッシング型通信を実行可能にする計算機間データ送受信方法を提供することである。

【0011】さらに、本発明の他の目的は、第1の通信網とそれより高速の第2の通信網に接続された計算機上で動作し、他のアプリケーションプログラムとの間で、たとえばTCP/IP通信規約のような第1の通信規約に基づいて通信するように構成されたアプリケーションプログラムが、第1の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間で、その通信規約に基づいた通信を実行することを可能にするとともに、第2の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間でその第2の通信網を使用した高速の通信を実行する行うことを可能にする計算機間データ送受信方法を提供することである。

【0012】さらに、本発明の他の具体的な目的は、上記第2の通信網を使用した通信を、メッセージパッシング型の通信とすることができる計算機間データ送受信方法を提供することである。

【0013】

【課題を解決するための手段】上記目的を達成するために、本発明による計算機間データ送受信方法では、第1の計算機上で実行されている第1のアプリケーションプログラムが発行した複数の送信命令により指定される複数の送信データのそれぞれを、第2の計算機上で実行されている第2のアプリケーションプログラムが発行した複数の受信命令に応答して、メッセージパッシング型の通信で受信する。この受信を第2の計算機上に設けられたエミュレーションライブラリにより制御する。

【0014】さらに、上記複数の送信データの連なりからなるひと繋がりデータの内、上記複数の受信命令により指定されるサイズ部分に区分して得られる部分をそれぞれの受信命令が指定する複数のバッファに格納するというストリーム通信を実現するように、受信されたそれぞれの送信データを処理する。この処理を、上記エミュレーションライブラリにより制御する。

【0015】より具体的には、本発明による計算機間データ送受信方法は、以下の処理を実行する。

【0016】(a) 送信側アプリケーションが一回の送信命令によって送信しようとするデータの長さを、受信側エミュレーションライブラリにより検知し、(b) 上記の送信データ長を、アプリケーションが一回の受信命令によって指定しているデータ受け取り長と受信側エミュレーションライブラリにより比較し、(c) 上記の比較で、送信データ長がデータ受け取り長より長ければ、受信側エミュレーションライブラリが、一旦メモリ上に確保したバッファ領域にデータの全てを受信して、そこからアプリケーションが指定したデータ受け取り長分だけのデータをアプリケーション領域にコピーし、送信データ長がデータ受け取り長以下であれば、受信側エミュレーションライブラリが、アプリケーション領域にデータを直接受信する。

【0017】(d) アプリケーションが受信命令を発行した時に、バッファ領域にデータが残っている場合には、そこからアプリケーション領域にデータをコピーする。

【0018】さらにより具体的には、本発明によるデータ送受信方法を実現するために、TCP/IPエミュレーションライブラリを用意する。このライブラリは、TCP/IPのソケットアプリケーションプログラムインタフェースと同一のインタフェースを持ち、通信相手が並列計算機の外部であつたら従来のシステムコールを、内部であつたらMPI等の並列計算機用メッセージパッシング型通信方式による高速通信網を用いる、という切り分けを行う。すなわち、このTCP/IPエミュレーションライブラリには以下のような特徴を持たせる。

【0019】(1) 並列計算機内部で通信する場合は、メッセージパッシング型通信方式に適した通信手順を用いて、TCP/IPと同等のストリーム通信サービスを提供する。

【0020】(2) TCP/IPエミュレーションライブラリとして適当な手段を用いて通信方法を切り分ける。

【0021】(3) 外部通信のデータと内部通信のデータをスピループによって検出する。もしくは外部通信のデータと内部通信のデータを別々のスレッドで検出する。

【0022】

【発明の実施の形態】<従来の技術とその問題点>本発明の実施の形態を説明する前に、従来の技術とその問題点を説明する。

【0023】(1) TCP/IP

図13にTCP/IPの階層を示す。TCP/IPでは、下からリンク層301、IP層302、TCP層303、アプリケーション層306の4階層を定めている。実際にはリンク層301の下に物理層が存在するが簡単化のためにこの層には言及しない。以下では、簡単化のためにアプリケーション層より下の層303、30

2, 301からなる層群304をまとめてTCP/IP層と呼ぶ。一般にTCP/IPが定める手順で通信を行なうために必要なプログラム（以下、TCP/IP処理ルーチンと呼ぶ）はOSに含まれる。TCP/IPが定める手順で通信を実行するには、TCP/IP層304を構成する複数の層の各々によりそれぞれ特定の処理が実行される。本明細書では、これらの層がそれぞれ実行する複数の処理あるいはそれらの処理を実行する複数のプログラムルーチンを総称してTCP/IP処理ルーチンと呼ぶ。

【0024】アプリケーションプログラムがTCP/IP処理ルーチンを利用するためのインタフェースとしては、一般にソケットアプリケーションプログラミングインタフェース（ソケットAPI）111が用いられる。アプリケーションプログラムがTCP/IP処理ルーチン（サーバ側）

```
sa=socket(AF_INET, SOCK_STREAM, 0);

bind(sa, server, slen);
listen(sa, 5);
sal=accept(sa, &client, &clen);
read(sal, buffer1, length1);
read(sal, buffer2, length2);

close(sal);
close(sa);
```

まず、他と通信を行おうとするサーバ側のTCP/IPアプリケーションプログラムおよびクライアント側のTCP/IPアプリケーションプログラムは、それが実行中の計算機に設けられたTCP/IP処理ルーチンに含まれたシステムコールソケット（socket）を呼び出す。呼び出されたシステムコールソケットは、通信端の役目を果たす、ソケットと呼ばれるオブジェクトを生成し、ソケット記述子を返す。上の例ではサーバ側、クライアント側のTCP/IPアプリケーションプログラムは、それぞれに返されるソケット記述子をsaまたはsb0として受け取る。

【0027】ソケット記述子は、アプリケーションプログラム内でソケット毎に一意に決定されるソケットの識別子（ID）であり、整数値からなる。生成されたソケットに対する操作は、全てこのソケット記述子を指定して行う。以下、特に断らない限り、ソケットとソケット記述子は同義であるとして説明する。上のシステムコールの引数AF\_INETは、アドレスファミリインターネットを表し、インターネットを介した通信にソケットを使用することを示す。さらに、引数SOCK\_STREAMは、ストリーム通信を要求する。最後の引数はプロトコルを指定する引数である。この値が0のときには、プ

ンを利用するために、ソケットライブラリと呼ばれるシステムコール群がOSの一部として設けられている。

【0025】なお、本明細書では、データ受信のためのシステムコールおよび関数あるいはデータ送信のためのシステムコールおよび関数を呼び出すことを、それぞれの機能と組み合わせて受信命令あるいは送信命令を発行すると記載することがある。ソケットアプリケーションプログラムインタフェースは、ソケットライブラリのアプリケーションプログラムインタフェースとして定められている。ソケットアプリケーションプログラムインタフェースに従って記述された、TCP/IPで通信を行なうアプリケーションプログラム（以下、TCP/IPアプリケーションプログラムと呼ぶ）の例を以下に示す。

【0026】（クライアント側）

```
connect(sb0, server, slen);
write(sb0, buffer0, length0);
write(sb0, buffer3, length3);

close(sb0);
```

ロトコルはその引数より前の二つの引数により決まる。今の場合には、TCP/IPプロトコルを使用することになる。ソケットが生成された後、サーバ側とクライアント側のTCP/IPアプリケーションプログラムは異なる手順でソケットの接続処理を行う。

【0028】サーバ側は、TCP/IP処理ルーチンに含まれたシステムコールバインド（bind）を呼び出す。呼び出されたシステムコールバインドは、引数で指定されたソケットsaに引数で指定された名前を対応付ける。名前は、IPアドレスとポート番号の組合わせからなる。上の例では、slenで指定される大きさを持つ構造体serverに格納されたIPアドレスとポート番号の組からなる名前にソケット識別子saが対応付けられる。サーバ側は、さらにTCP/IP処理ルーチンに含まれたシステムコールリスン（listen）を呼び出す。

【0029】このシステムコールリスンは、引数で指定されたソケットsaを接続要求の受領のためのソケットに設定する。このシステムコールの第2の引数は、その接続要求および他の接続要求を一時的に保持するのに用いるキューに要求するサイズを表し、今の場合には、このコールは、5個の接続要求を保持可能なキューを要求



している。

【0030】以上の処理の結果、論理的には、サーバ側とクライアント側の間に通信路が決定されたことになる。

【0031】その後、サーバ側は、TCP/IP処理ルーチンに含まれたシステムコールアクセプト (accept) を呼び出す。このシステムコールは、引数により指定されたソケット *s a* を接続要求の待ち状態にする。このシステムコールの第2、第3の引数は、待つべき接続要求の発行元クライアント側のIPアドレスと長さを表す。

【0032】一方、クライアント側は、TCP/IP処理ルーチンに含まれたシステムコールコネクト (connect) を呼び出す。このシステムコールは、引数で指定された名前、今の場合にはサーバ側のソケット *s a* に付けられた名前 (構造体 *server* に格納された、IPアドレス *sin\_addr* とポート番号 *sin\_port* の組み合わせ) に対して引数で指定されたソケット *s b 0* を接続する。

【0033】さらに、サーバ側の計算機では、先に呼び出されたシステムコールアクセプトが、この接続要求を受信し、システムコールアクセプトに対する前述の引数が指定するソケット *s a 1* をこのクライアント側との通信のためのソケットとして新たに生成する。こうして、サーバ側のソケット *s a 1* とクライアント側のソケット *s b 0* との間で通信路が確立されたことになる。

【0034】ソケット間の接続が確立された状態でクライアント側がサーバ側にデータを送信する時には、クライアント側は、TCP/IP処理ルーチンに含まれたシステムコールライト (write) を呼び出す。このシステムコールに対する引数では、送信すべきデータを保持するのに用いるバッファのアドレス *buffer 0* とそのバッファの長さ *length 0* を指定する。システムコールライトは、このバッファ内のデータをこのシステムコールが指定するソケット *s b 0* を用いて送信する。サーバ側はTCP/IP処理ルーチンに含まれたシステムコールリード (read) を呼び出す。このシステムコールは、その引数で指定されたソケット *s a 1* を介して転送されたデータを受信し、引数で指定されたアドレス *buffer 1* のバッファに書き込む。このようにして、2つのアプリケーションプログラム間でデータが転送される。もし必要ならば、クライアント側は複数の後続のデータをそれぞれ送信するために複数のライトシステムコールを呼び出し、サーバ側はそれらの後続のデータを受信するために一つまたは複数のリードシステムコールを呼び出す。

【0035】ソケット記述子は、ファイルへの入出力や標準入出力などを行う際に用いられる、ファイル記述子の一種として定義されている。このためソケットアプリケーションプログラムインタフェースでは、ファイル等

の入出力を行う場合と同様のインタフェースを通して、データの送受信を行うことができる。

【0036】サーバ側とクライアント側のプログラムの間の通信が終了したときには、サーバ側は、システムコールクローズ (close) を呼び出し、引数で指定されたソケット *s a 1* を閉鎖する。サーバ側は、さらに同じシステムコールクローズを再度呼び出し、システムコールクローズは今度はソケット *s a* を閉鎖する。クライアント側も同様にシステムコールクローズを呼び出して、このシステムコールはソケット *s b 0* を閉鎖する。

【0037】(2) 高速ソケットにおける通信方法の切り替え

従来のTCP/IPでは、システムコールバインドによってIPアドレスとポート番号からなる名前 (server) をソケット (s a) に対応付けていた。高速ソケットでは、この時、さらに高速通信専用のソケットを以下のように生成する。まず、アプリケーションがシステムコールバインドの呼び出し時に指定する名前のポート番号 (構造体 *server* に格納されたポート番号 *sin\_port*) にハッシュ関数を施して、新たにシャドウポート番号を導き出す。次に、システムコールソケットを呼び出して、シャドウソケットを新規に生成する。最後にバインドシステムコールを呼び出して、シャドウソケットにシャドウポート番号を対応付ける。高速通信を行なう場合は、サーバおよびクライアントがacceptおよびconnectを呼び出して接続するときに、このシャドウソケットを用いることで高速通信専用の通信路を貼る。

【0038】この方法では、システムコールバインドやシステムコールリスンが呼び出される時に、特別な処理が必要となる。

【0039】(3) TCP/IP処理ルーチンによるストリーム通信

TCP/IPのソケットライブラリは、ストリーム通信をサービスする。ストリーム通信とは、送信側のアプリケーションプログラムが一連のライトシステムコールにより送る複数のデータをひと繋ぎりのデータストリームとして処理し、そのデータストリームを受信側のアプリケーションプログラムが発行する一つあるいは複数のリードシステムコールが指定する任意の長さを有する一つあるいは複数のデータに切り分けて受け取るという通信方法である。

【0040】図14を用いて従来のストリーム通信の動作を説明する。ここでは、送信アプリケーションプログラム801が受信アプリケーションプログラム802にデータを送る例を示している。送信アプリケーションプログラム801は、まず第1のライトシステムコールを呼び出してそのプログラムのバッファ803内の50キロバイト (KB) のデータを送信する (805)。なお、図14では単位KBは簡単化のために省略してい

る。送信側のTCP/IP処理ルーチンは、バッファ803内のこの送信データを一旦OS内に確保したバッファ（図示せず）にコピーし、複数のパケットに分割して、受信側OSに送信する。通常パケットのサイズは40～1500バイトである。受信側OSは、それらのパケットをOS内に確保した複数のバッファ（図示せず）に受け取り、これらのパケットをリスト状に繋いで、データストリームを再構成する。送信アプリケーションプログラム801は、さらに第2のライトシステムコールを発行してそのプログラムの他のバッファ804内の80KBのデータを送信する（806）。このときも送信側のTCP/IP処理ルーチンと受信側のTCP/IP処理ルーチンは同様に動作して、このデータを先に送信されたデータと組み合わせて一つのデータストリームとしてOS内の前述のバッファに保持する。

【0041】図において、809はOS900内に保持されたこのデータストリームを模式的に表す。先頭のデータ807、後続のデータ808はそれぞれ上記第1、第2のライトシステムコールにより送信されたそれぞれ50KB、80KBのデータを表す。ストリーム通信では、受信アプリケーションプログラムは、これらのデータを一本の連続した130KBのデータストリーム809として捉える。送信アプリケーションプログラム801内のバッファ803、804内のデータは、OS900内のデータストリーム809内のデータ部分807、809をそれぞれ保持する。このための、受信アプリケーションプログラム802は、第1のリードシステムコールを発行して、ストリームデータ809内の50KBの先頭データ807の内の30KBを受信アプリケーションプログラム802の30KBのバッファ812に受け取ることを要求する（814）。受信側のTCP/IP処理ルーチンは、このリードシステムコールにตอบสนองして、このデータストリーム809から先頭の30KBのデータ810をバッファ812にコピーする。受信アプリケーションプログラム802がさらに第2のリードシステムコールを発行すると、このシステムコールが指定する長さに従って、ストリームデータ809内にある残りの100KBのデータ811をこのシステムコールが指定するバッファ813にコピーする（815）。

【0042】(4) アクティブメッセージ型通信でのストリーム通信

上記高速ソケットを使用したシステムでは、アクティブメッセージ型の通信を採用しながらストリーム通信を実現している。すなわち、このシステムでは、送信側のアプリケーションプログラムから送信要求が発行されると、この送信要求は、受信側のアプリケーションプログラムから受信要求が発行されるのを待たないで実行される。送信要求は、受信側の計算機に割り込みを発生し、この割り込みにより割り込みハンドラーが起動され、送信データは、この割り込みハンドラー内のバッファに一

旦受信される。受信側のアプリケーションプログラムが受信要求を発行すると、割り込みハンドラーは、既に受信されたデータの内、この受信要求が要求する大きさのデータを、受信側のアプリケーションプログラムのバッファに転送する。もし、受信側のアプリケーションプログラムが要求するサイズが、受信済みのデータのサイズより小さければ、受信要求の処理が終了する。割り込みハンドラーに保持された残りのデータは、受信側のアプリケーションプログラムから新たな受信要求が発行されたときに、そのアプリケーションプログラムのバッファに転送される。

【0043】逆に、受信側のアプリケーションプログラムが要求するサイズが、受信済みのデータのサイズより大きければ、割り込みハンドラーは、送信側のアプリケーションプログラムがその後データを送信してきたときに、そのデータを、受信側のアプリケーションプログラムに供給する。受信側の割り込みハンドラーは、送信側のアプリケーションプログラムが送信要求を発行する前に、受信側のアプリケーションプログラムから後続の受信要求が発行されたときにも、同様にその後送信側のアプリケーションプログラムが後続のデータをその後送信してきたときに、その後続のデータを受信側のアプリケーションプログラムに供給する。もし、この後続のデータが、上記最初の受信要求が要求する不足のデータのサイズあるいは上記後続の受信要求が要求するデータのサイズより大きければ、割り込みハンドラーは、あまりのデータをさらに後続の受信要求のために保持する。

【0044】こうして、送信側のアプリケーションプログラムが発行する複数の送信要求により送信される複数のデータを、受信側のアプリケーションプログラムが発行する一連の受信要求にตอบสนองして受信側のアプリケーションプログラムに供給する。こうして、この方法では、割り込みハンドラー内のバッファに送信データを一旦保持することにより、ストリーム通信を実現している。

【0045】しかし、割込を使用しないメッセージパッシング型の通信では、このような方法を使用してストリーム通信を実現できない。

【0046】以下、本発明に係る計算機間データ送受信方法を図面に示したいくつかの実施の形態を参照してさらに詳細に説明する。なお、以下においては、同じ参照番号は同じものもしくは類似のものを表わすものとする。また、発明の第2の実施の形態以降においては、発明の第1の実施の形態との相違点を主に説明するに止める。

#### 【0047】＜発明の実施の形態1＞

##### (1) 装置の概要

図1は、本発明に係る計算機間送受信方法を実行するための計算機システムの一例を示す。図において、並列計算機101の内部の2台の計算機102、103と、1台の外部計算機104とがお互いに通信網で繋がってい

ると仮定する。実際には、並列計算機 101 の内部および外部の計算機の台数はそれぞれ任意である。内部の計算機 102 と 103 は内部高速通信網 105 で繋がっており、内部の計算機 102, 103 にはそれぞれ内部高速通信網 105 専用のネットワークインタフェースハードウェア 119, 120 が存在する。内部高速通信網 105 は、複数のバケットを互いに並列にかつ高速に転送可能なネットワーク、たとえばハイパクロスバススイッチなどにより構成される。また、内部の計算機 102, 103 の全てと外部の計算機 104 はグローバルな通信網 106 に繋がっており、各計算機にはそれぞれ通信網 106 専用のネットワークインタフェースハードウェア 121, 122, 123 が存在する。

【0048】データの送受信は、各計算機のメモリ 124, 125, 126 にロードされているアプリケーションプログラム 107, 108, 109 の間で行われる。また、各メモリにはアプリケーションプログラムの他に OS 127, 128, 129 がロードされており、それぞれの OS の中には TCP/IP 処理ルーチン 114, 115, 110 が存在する。TCP/IP が定める手順で通信を実行するには、TCP/IP 層 304 を構成する各層によりそれぞれ処理が実行される。TCP/IP 処理ルーチンは、TCP/IP が定める手順で通信を実行するためにこれらの層がそれぞれ実行する複数の処理の総称で、これらの TCP/IP 処理ルーチン 114, 115, 110 自体は公知のものと同一であり、既に述べたような、システムコールで呼び出し可能な複数の関数を含んでいる。TCP/IP 処理ルーチン 114, 115, 110 は、広域通信を目的として、グローバルな通信網 106 専用のネットワークインタフェースハードウェア 121, 122, 123 によって通信を行う。

【0049】計算機 102 のメモリ 124 には、さらに TCP/IP エミュレーションライブラリ 112 とメッセージパッシング型ライブラリ 140 と高速通信ライブラリ 135 とがロードされている。同様に、計算機 103 のメモリ 125 には、TCP/IP エミュレーションライブラリ 113 とメッセージパッシング型ライブラリ 141 と高速通信ライブラリ 136 とがロードされている。

【0050】計算機 102, 103 上の高速通信ライブラリ 135, 136 は並列計算機 101 の内部での高速通信を目的として、内部高速通信網 105 専用のネットワークインタフェースハードウェア 119, 120 によって通信を行うためのライブラリであり、リモートメモリ書き込みライブラリあるいは PUT ライブラリと呼ばれているものが多く使用される。本実施の形態でも高速通信ライブラリ 135, 136 にはこの PUT ライブラリを使用する。しかし、本発明はこのライブラリに限定されるのではなく、他のライブラリたとえば PUT/GET ライブラリと呼ばれるライブラリも使用可能であ

る。

【0051】一般に、高速内部通信網 105 を通した通信はグローバルな通信網 106 を通して通信する場合に比べて格段に速い。そこで TCP/IP エミュレーションライブラリ 112, 113 は、内部計算機上のアプリケーションプログラム同士が TCP/IP 通信を行おうとする際には、TCP/IP 処理ルーチン 114, 115 ではなくメッセージパッシング型ライブラリ 140, 141 と高速通信ライブラリ 135, 136 と内部高速通信網 105 を使用してメッセージパッシング型の通信を実現するように構成され、それでもって通信の高速化を図る。

【0052】メッセージパッシング型ライブラリ 140, 141 は、TCP/IP エミュレーションライブラリ 112 または 113 からの要求にしたがって、高速通信ライブラリ 135 または 136 を起動するためのライブラリである。メッセージパッシング型ライブラリ 140, 141 は、一般的にはアプリケーションプログラム（本実施の形態においては TCP/IP エミュレーションライブラリ 112, 113）に対してメッセージパッシング型のインタフェースを有するライブラリである。

【0053】TCP/IP エミュレーションライブラリ 112, 113 は、さらに、このメッセージパッシング型の通信においても従来の TCP/IP 処理ルーチンが提供していたのと同じくストリーム通信を実現し、それでもって通信の高速化を図る。TCP/IP 処理ルーチン 114, 115 は、OS 127, 128 の機能であるため、従来技術ではアプリケーションプログラムがこれらを利用する際には必ずコンテキストスイッチのオーバーヘッドが発生するが、本実施の形態では、高速通信ライブラリ 135, 136 を使うので、OS を介さないためオーバーヘッドを回避でき、それにより、より一層の通信の高速化も期待できる。

【0054】

(2) 論理構成と構成要素間のインタフェース図 1

(B) は、上記ハードウェア構成を論理構成として表現した図である。この図では、以降の説明に関係の無い部分は全て省略してある。また、高速通信ライブラリ 135, 136 と内部高速通信専用ハードウェア 119, 120 をひとまとめにして高速通信機構 116, 117 と表現している。さらに、各構成要素間のインタフェース 111 および 118 を新たに示している。計算機 102, 103, 104 や並列計算機 101 を表す四角はそれぞれ、その中の論理構成要素が一台の計算機または並列計算機 101 上で実行されることを示している。

【0055】計算機 104 上では従来通り、アプリケーションプログラム 109 が TCP/IP 110 と、ソケットアプリケーションプログラムインタフェース 111 でリンクされている。並列計算機 101 の内部の計算機 102, 103 上で動くアプリケーションプログラム 1

07, 108は、TCP/IPエミュレーションライブラリ112, 113にソケットアプリケーションプログラムインタフェース111でもってリンクされている。また、TCP/IPエミュレーションライブラリ112, 113は、OSの機能である従来のTCP/IP処理ルーチン114, 115にソケットアプリケーションプログラムインタフェース111でもってリンクされ、同時に、高速通信機構116, 117にMPI仕様のインタフェース118でもってリンクされている。

【0056】(3)アプリケーションプログラム107, 108

本実施の形態では、並列計算機101内のいずれかの計算機102上で動作しているアプリケーションプログラム例えば107が、いずれかの他の計算機上で動作している他のアプリケーションプログラムと通信する場合、当該他のアプリケーションプログラムが、並列計算機101内の計算機103上で動作しているアプリケーションプログラム例えば108か並列計算機101外の計算機104上で動作しているアプリケーションプログラム例えば109かによって、高速通信機構116とTCP/IP処理ルーチン114を使い分ける。並列計算機101内の異なる計算機102, 103上で動作する二つ

(サーバ側)

```
sa=socket(AF_INET, SOCK_STREAM, 0);
```

```
bind(sa, server, slen);
```

```
listen(sa, 5);
```

```
sal=EMU_accept(sa, &client, &crlen);
```

```
EMU_read(sal, buffer1, length1);
```

```
EMU_read(sal, buffer1, length1);
```

```
.
```

```
.
```

```
.
```

```
close(sal);
```

```
close(sa);
```

サーバ側アプリケーションプログラムは従来と同様にsocket, bind, listenに対するシステムコールを呼び出す。これらのシステムコールは対応するTCP/IP処理ルーチンにより従来と同様に処理される。クライアント側のアプリケーションプログラムも従来と同様にsocketシステムコールを発行する。このシステムコールも対応するTCP/IP処理ルーチンにより従来と同様に処理される。こうしてサーバ側とクライアント側に対して従来と同様にソケットsa, sb0が生成される。

【0059】その後サーバ側アプリケーションプログラムは、従来のシステムコールアクセプトに代えて、そのアプリケーションプログラムが動作している計算機内に設けられたTCP/IPエミュレーションライブラリ内に設けられた関数エミュレーションアクセプト(EMU

のアプリケーションプログラム107, 108が高速通信機構116, 117を使用して相互に通信するためには、それらのアプリケーションプログラム間でデータを実際に送受信するときだけでなく、それぞれのアプリケーションプログラムのためのソケットを相互に接続するときにも高速通信機構116, 117を利用するように特別の処理を行う必要がある。TCP/IPエミュレーションライブラリ112または113がこの特別の処理を実行するための複数の関数を含む。本実施の形態では、TCP/IPエミュレーションライブラリ112, 113内に設けられた関数の名前にEMU\_という接頭辞をつけ、TCP/IP処理ルーチン114, 115または110内に設けられた前述の関数の名前と区別する。

【0057】具体的には、並列計算機101内の計算機102, 103上で動作しているアプリケーションプログラム107または108の内、サーバ側およびクライアント側として動作するアプリケーションプログラムはそれぞれ以下のプログラムを実行するように生成される。

【0058】

(クライアント側)

```
sb0=socket(AF_INET, SOCK_STREAM, 0);
```

```
EMU_connect(sb0, server, slen);
```

```
EMU_write(sb0, buffer0, length0);
```

```
EMU_write(sb0, buffer0, length0);
```

```
.
```

```
.
```

```
.
```

```
close(sb0);
```

\_\_accept)を呼び出す。クライアント側のアプリケーションプログラムは、従来の関数コネクต์に代えて、そのアプリケーションプログラムが動作している計算機内に設けられたTCP/IPエミュレーションライブラリ内に設けられた関数エミュレーションコネクต์(EMU\_connect)を呼び出す。さらに、サーバ側アプリケーションプログラムは、従来のシステムコールリードに代えて、TCP/IPエミュレーションライブラリ内に設けられた関数エミュレーションリード(EMU\_read)を呼び出し、クライアント側アプリケーションプログラムは、従来のシステムコールライトに代えて、対応するTCP/IPエミュレーションライブラリ内に設けられた関数エミュレーションライト(EMU\_write)を呼び出す。以下、これらの新たな関数が行う処理を説明する。

【0060】(4) TCP/IPエミュレーションライブラリによるソケットの接続

図2において、サーバ側アプリケーションプログラム、クライアント側アプリケーションプログラムが、それぞれ上述のEMU\_\_accept, EMU\_\_connect関数を呼び出すと(処理501, 502)、これらのシステムコールによってサーバ側のTCP/IPエミュレーションライブラリ内に設けられた関数EMU\_\_acceptとクライアント側のTCP/IPエミュレーションライブラリ内に設けられたEMU\_\_connect呼び出される。これらのシステムコールの引数は、サーバ側アプリケーションプログラム、クライアント側アプリケーションプログラムが、それぞれシステムコールaccept, connectを呼び出し、ソケットsa, sb0の接続をTCP/IP処理ルーチン114, 115に要求するときと同じである。

【0061】呼び出された関数EMU\_\_acceptとEMU\_\_connectは、まずacceptシステムコール, connectシステムコールをそれぞれ発行する(処理503, 504)。これらのシステムコールの引数は、関数EMU\_\_accept, EMU\_\_connectのそれぞれに対する引数がそのまま使用される。これによりサーバ側のTCP/IP処理ルーチン内に設けられたシステムコールacceptとクライアント側のTCP/IP処理ルーチン内に設けられたシステムコールconnectが呼び出され、従来と同様にコールされたシステムコールacceptはコールされたシステムコールconnectによって発行された接続要求を受領し、ソケットsa1が生成され、ソケットsa1とソケットsb0が通信路106を介して接続された状態になる。

【0062】その後、関数EMU\_\_accept, EMU\_\_connectは、それぞれ相手のIPアドレスが並列計算機101の内部の計算機のアドレスであるか否かを確認する(処理506, 507)。相手が並列計算機101の内部である場合は、ソケット記述子を内部テーブル410または411に登録する(処理510, 511)。今の場合には、サーバ側のアプリケーションプログラム107用のソケットsa1、クライアント側のアプリケーションプログラム用のソケットsb0がそれぞれ内部テーブル410, 411にそれぞれ登録される。さらに、TCP/IPエミュレーションライブラリ112, 113は、並列計算機101内における各計算機の識別子などの高速通信機構116, 117を用いるために必要なデータを交換する。このデータ交換のために、アプリケーションプログラム107, 108はそれぞれ関数read, writeシステムコールを発行する(処理512, 513, 515, 516)。ここでのデータ交換は、既に接続されているソケットsa1とsb0を利用し、従来と同様にTCP/IP処理ルーチン

114, 115と通信路106を介して行われる。その後それぞれサーバ側、クライアント側のアプリケーションプログラム107, 108にリターンする(処理518, 519)。このとき、サーバ側のTCP/IPエミュレーションライブラリ112はアプリケーションプログラム107に生成されたソケットsa1の識別子を戻す。

【0063】処理506, 507において、相手が並列計算機101の内部の計算機でないことが判明したときは、関数EMU\_\_connectはクライアント側のアプリケーションプログラム108に返り、関数EMU\_\_acceptはソケットsa1をサーバ側のアプリケーションプログラム107に返し、そのプログラムに戻る(処理508, 509)。こうして、サーバ側のアプリケーションプログラムに対するソケットsa1とクライアント側のアプリケーションプログラムに対するソケットsb0は、それぞれに対応するTCP/IP処理ルーチンと通信路106を介して接続される。

【0064】なお、図3は、本実施の形態によりアプリケーションプログラム107, 108, 109が生成したソケットがお互いに接続されている一つの状態を示している。ここでは、アプリケーションプログラム107のソケットSA1(402)とアプリケーションプログラム108のソケットSB0(403)とが接続され(407)、アプリケーションプログラム108のソケットSB1(404)とアプリケーションプログラム109のソケットSC0(405)が接続され(408)、アプリケーションプログラム109のソケットSC1(406)とアプリケーションプログラム107のソケットSA0(401)が接続されている(409)。ここで、TCP/IPエミュレーションライブラリ112, 113は、内部ソケットテーブル410, 411を保持している。内部ソケットテーブル410, 411には、ソケットの接続先が内部計算機である場合に、そのソケット記述子を登録する。例えば、アプリケーションプログラム107のソケットSA1(402)の接続先であるソケットSB0(403)は、内部計算機103の上で動くアプリケーションプログラム108のソケットなので、SA1を内部ソケットテーブル410に登録する。同様に、ソケットSB0の接続先であるソケットSA1は、内部計算機102の上で動くアプリケーションプログラム107のソケットなので、SB0を内部ソケットテーブル411に登録する。一方、ソケットSA0(401)の接続先であるソケットSC1(406)は、外部計算機上アプリケーションプログラム109のソケットなので、SA0は内部ソケットテーブル410には登録しない。同様に、SC0に接続されているSB1も内部ソケットテーブル411には登録しない。

【0065】(5) 内部通信と外部通信の切り分け方法  
その後、サーバ側のアプリケーションプログラムとクラ

クライアント側のアプリケーションプログラムはデータ通信を開始する。これらの二つのプログラムの内的一方および他方は、データ送信および受信のために関数EMU\_write, EMU\_readをそれぞれ呼び出す。先に示したクライアント側とサーバ側のプログラムの例では、サーバ側のアプリケーションプログラムが、関数EMU\_readを呼び出し、クライアント側のアプリケーションプログラムが関数EMU\_writeを呼び出している。これらで指定する引数は、TCP/IP処理ルーチン114, 115に含まれたシステムコールwrite, readに対する引数と同じである。

【0066】図4を参照するに、EMU\_read, EMU\_writeが呼び出されると（処理701, 702）、それぞれの関数は、それぞれの引数で指定されたソケットsa1, sb0のソケット記述子が対応する内部ソケットテーブル410, 411にそれぞれ登録されているか否かを判定する（処理703, 704）。それぞれのソケット識別子が内部ソケットテーブル410, 411にそれぞれ登録されているときには、後に詳細に説明する手順で高速通信機構116, 117、高速内部通信網105を用いてメッセージパッシング方式の通信を行う（処理707, 708）。それぞれのソケット識別子が内部ソケットテーブル410, 411に登録されていないかったら、そのまま従来のread, writeシステムコールをそれぞれ指定されたソケットに対して発行する（処理705, 706）。これらのシステムコールは対応するTCP/IP処理ルーチン114, 115により処理され、グローバル通信路106を用いた通信がそれらのTCP/IP処理ルーチン114, 115により実行される。

【0067】なお、図5は、図3で示したソケットの接続状態において、各アプリケーションプログラム107, 108, 109同士が通信を行う場合のデータの流れを示して、上記切り分け方法を説明するための全体構成図である。アプリケーションプログラム107とアプリケーションプログラム108が通信する場合は、ソケットsa1（402）とソケットsb0（403）を通信端として用いる。これらのソケットは、TCP/IPエミュレーションライブラリ112, 113が保持する内部ソケットテーブル410, 411に登録されている。そこでTCP/IPエミュレーションライブラリ112, 113は、データ通信処理にTCP/IP処理ルーチン114, 115ではなく高速通信機構116, 117を利用する（601）。一方、アプリケーションプログラム108とアプリケーションプログラム109が通信する場合は、ソケットSB1（404）とソケットSC0（405）を通信端として用いる。アプリケーションプログラム108とリンクされているTCP/IPエミュレーションライブラリ113が保持する内部ソケットテーブル411には、SB1は登録されていない。

そこでTCP/IPエミュレーションライブラリ113は、データ通信処理に、TCP/IP処理ルーチン115をそのまま利用する（602）。これによって、外部のTCP/IP処理ルーチン110とのデータ交換が可能となる。

【0068】本方式により、バインドシステムコール、リスンシステムコールは従来のTCP/IP処理ルーチンから変更せずにそのまま用いて、内部通信と外部通信の切り分けを実現できる。

【0069】（6）メッセージパッシング型ライブラリ140, 141

並列計算機の各計算機に使用される内部高速通信網105を使用するための通信ハードウェア119, 120および高速通信ライブラリ135, 136はベンダ特有である場合が多いので、その利用方法もマシンによって様々である。よって、各マシンの高速通信ハードウェアと高速通信ライブラリを利用したアプリケーションプログラムを作ろうとする場合、マシンに特化した汎用性の低いプログラムにならざるを得なかった。これに対して、並列計算機内部の通信ハードウェアを利用して通信するためのライブラリを用意し、このライブラリのアプリケーションプログラムインタフェースを標準として規定することで、アプリケーションプログラムの汎用性を高めようという活動が世界中で活発である。

【0070】このメッセージ型の通信を使用するための汎用のインタフェースとして現在広く使用されているインタフェースは、MPIと呼ばれるメッセージパッシングインタフェース（MPI—Message Passing Interface）である。例えば、文献：“MPI: Message Passing Interface Standard version 1.1”, MPI Forum, University of Tennessee, 1995参照。このインタフェースは、メッセージパッシング型の通信を実現するための上に述べた高速通信ライブラリが存在することを前提としているものであり、このインタフェースを使用しても、メッセージパッシング型の通信は、基本的には上記高速通信ライブラリにより実現されることには変わらない。

【0071】多くの並列計算機ベンダが、MPIに準拠したアプリケーションプログラムが並列計算機内部の高速通信ハードウェアを使用できるようにするための高速通信ライブラリを提供している。

【0072】本明細書では、メッセージ型の通信を使用するためのインタフェースをメッセージパッシング型インタフェースと呼び、そのインタフェースを有するライブラリをメッセージパッシング型ライブラリと呼ぶ。特に、MPI仕様のインタフェースをMPIあるいはMPIインタフェースあるいはメッセージパッシングインタフェースと呼び、そのインタフェースを有するライブラリをMPIライブラリあるいはメッセージパッシングインタフェースライブラリと呼ぶことがある。

【0073】本実施の形態では、メッセージパッシング型ライブラリ140、141として多くの並列計算機で利用可能である標準のMPI仕様により定められたインタフェースでもってコマンドあるいはデータを交換するライブラリを使用する。それでもって、TCP/IPエミュレーションライブラリ112、113の汎用性を高める。しかし、他の仕様のインタフェースを使用してもよい。

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) /* receiver */
    MPI_Recv(buffer, length, type, 1, tag, comm, status);
else if (rank == 1) /* sender */
    MPI_Send(buffer, length, type, 0, tag, comm);
MPI_Finalize();
```

MPIでは、通信しようとする全プロセスを一斉に立ち上げる。この時、各プロセスにはランクと呼ばれるプロセス識別子が決定される。上の例では、2つのプロセス、すなわち送信側プロセスsenderと受信側プロセスreceiverとを立ち上げる場合を示している。この時、それぞれのプロセスにはランク0と1が付けられる。各プロセスは、まずMPI初期化関数MPI\_InitによってMPIの初期化を行い、MPI通信ランク関数MPI\_Comm\_rankによって自分のランクを取得する。その後、各ランクごとの処理を行う。上の例では、ランク1のプロセスがMPI送信関数MPI\_Sendによって送るデータを、ランク0のプロセスがMPI受信関数MPI\_Recvで受け取る場合を示している。

【0076】(7) 高速通信機構を用いたメッセージパッシング型の通信

高速通信機構116、117を用いるデータ受信処理707、データ送信処理708の詳細を説明する前に、これらの処理の実行に必要な、高速通信機構116、117を用いたメッセージパッシング型の通信の概要を説明する。

【0077】一般にメッセージパッシング型の通信を実行するには、送信側の計算機がデータを送信するための送信関数を実行し、受信側の計算機がそのデータを受信するための受信関数を実行する。後続のデータをそれらの計算機の間で転送するために、それらの関数が繰り返し実行される。本実施の形態では、メッセージパッシング型の通信の具体的な説明として、MPI仕様によるメッセージパッシング通信を説明する。

【0078】MPI仕様によるメッセージパッシング通信では、送信側の計算機は送信関数MPI\_Sendを実行し、受信側の計算機は受信関数MPI\_Recvを実行する。今の場合、送信処理708が起動されると、送信側のTCP/IPエミュレーションライブラリ113は、先に記載したMPI仕様のアプリケーションプロ

【0074】MPIに準拠した従来のアプリケーションプログラムの記述例を以下に示す。本実施の形態では、TCP/IPエミュレーションライブラリ112、113は、メッセージパッシング型ライブラリ140、141を起動する部分に関しては、以下に示すプログラム部分を有する。このプログラム部分のより詳細は、後にふれる。

【0075】

グラムの例にあるように、送信関数MPI\_Sendを呼び出す。なお、送信関数MPI\_Sendを呼び出す前に初期処理としてMPI\_Init、MPI\_Comm\_rank等の関数の呼び出しを実行する必要があるが、これらの処理は、先に記載した関数EMU\_accept、EMU\_connect内の内部テーブル登録処理(510、511)と同時に進行する。一方、受信側のTCP/IPエミュレーションライブラリ112は、内部高速通信網105を用いるために受信関数MPI\_Recvを呼び出す。

【0079】送信側のTCP/IPエミュレーションライブラリ113が上記送信処理708内で呼び出す関数MPI\_Sendの引数で指定するバッファアドレスbufferおよびバッファ長lengthは、送信側のアプリケーションプログラム108が発行した関数EMU\_writeの対応する引数に等しくされる。一方、受信側のTCP/IPエミュレーションライブラリ112が受信処理707内で呼び出す関数MPI\_Recvの引数で指定するバッファアドレスbufferおよびバッファ長lengthは、後に説明するように受信側のアプリケーションプログラム107が発行した、関数EMU\_readの引数が指定するバッファアドレスおよびバッファ長もしくは受信側のTCP/IPエミュレーションライブラリ113内に設けられる特定のバッファのアドレスおよびサイズに等しくされる。

【0080】送信側のメッセージパッシング型ライブラリ141内の関数MPI\_Sendは、起動されると、対応する高速通信ライブラリ136に対し、関数MPI\_Sendに対する引数が指定するアドレスのバッファからその引数が指定する長さのデータを内部高速通信網105を介して受信側の計算機124に転送することを要求する。受信側のメッセージパッシング型ライブラリ140内の関数MPI\_Recvは、起動されると、対応する高速通信ライブラリ135に対し、その高速通信ライブラリ135に、この転送されたデータを内部高速



通信網 1 0 5 を介して受信し、関数 `MPI_Recv` に対する引数が指定するアドレスのバッファにその引数が指定する長さのデータを書き込むことを要求する。

【0081】起動された送信側の高速転送ライブラリ 1 3 6 と受信側の高速転送ライブラリ 1 3 5 は、要求されたデータの送信と受信をそれ自体公知の方法により内部高速通信網 1 0 5 を介して行う。このデータ転送は具体的には以下のように行われる。一般にはリモートメモリ書き込みコマンドあるいは `PUT` コマンドといわれるコマンドが使用される。以下では、このコマンドを `PUT` コマンドと呼ぶ。内部高速通信網 1 0 5 を介したデータ転送を行う通信方法は 3 つの `PUT` コマンドにより行われる。

【0082】まず、送信側の `TCP/IP` エミュレーションライブラリ 1 1 3 から関数 `MPI_Send` が呼び出されると、送信側のメッセージパッシング型ライブラリ 1 4 1 は、この引数が指定したバッファ長（これはすなわち送信すべきデータの長さである）を含むデータの属性情報を含むヘッダの送信を送信側の高速転送ライブラリ 1 3 6 に要求する第 1 の `PUT` コマンドを発行する。この高速転送ライブラリ 1 3 6 は、受信側の計算機 1 0 2 にこのヘッダを送信する。計算機 1 0 2 内の内部高速通信専用ハードウェア 1 1 9 が、このデータを受信側のメモリ 1 2 4 内の所定の位置に直接書き込む。

【0083】一方、受信側 `TCP/IP` エミュレーションライブラリ 1 1 2 から受信命令 `MPI_Recv` が呼び出されると、受信側のメッセージパッシング型ライブラリ 1 4 0 は、まず送信側の計算機 1 0 3 からヘッダがすでに送られてきているか調べる。このために後述するデータ検査命令 `MPI_probe` が使用される。ヘッダが送られてきていないときには、その受信命令の実行は終了する。ヘッダがすでに送られてきているときには、そのヘッダ内のバッファ長から送信データが受信可能か否かを判定する。すなわち、その送信データの長さが、上記受信命令 `MPI_Recv` が指定したバッファ長以下であるか否かが判定される。`MPI` 仕様のメッセージパッシング通信に限らず、一般にメッセージパッシング型の通信では、送信命令が指定した送信データ長が、受信命令が指定した受信バッファサイズより大きいときには、受信側の計算機はその送信データを受信しないようになっている。もし受信側の計算機が受信側のバッファのサイズを超える受信データを受信したときには、そのバッファ以外のメモリ領域が受信データにより破壊する恐れがあるためである。受信側のメッセージパッシング型ライブラリ 1 4 0 は、上記判定の結果、送信データが受信可能であると判断したときには、受信命令 `MPI_Recv` に対する引数が指定した受信バッファのアドレスの送信を受信側の高速通信ライブラリ 1 3 5 に要求する第 2 の `PUT` コマンドを発行する。この高速通信ライブラリ 1 3 5 はこのコマンドに対する応答とし

てバッファアドレスを送信側の計算機 1 0 3 に送信する。1 0 2 内の内部高速通信専用ハードウェア 1 1 9 は、このアドレスを送信側のメモリ 1 2 5 内の所定の位置に書き込む。なお、上記判定の結果、送信データが受信可能でないときには、受信側のメッセージパッシング型ライブラリはバッファアドレスを返さないで、エラーを送信側の計算機に通知する。

【0084】最後に、送信側のメッセージパッシング型ライブラリは、送信データと受信されたバッファアドレスの送信を送信側の高速転送ライブラリ 1 3 6 に要求する第 3 の `PUT` コマンドを発行する。高速転送ライブラリ 1 3 6 は、このデータとバッファアドレスを受信側の計算機 1 0 2 に送信する。受信側の計算機 1 0 2 内の内部高速通信専用ハードウェア 1 1 9 は、このバッファアドレスとデータを受信し、そのアドレスを有するバッファに受信したデータを直接書き込む。こうして、データの転送が終了する。

【0085】なお、内部高速通信網 1 0 5 を介したデータ通信方法は以上の方法に限らず他の方法も使用可能である。たとえば並列計算機によっては、高速通信ライブラリは `PUT` コマンドの他にリモートメモリ書き込みコマンドあるいは `GET` コマンドも実行可能である。このような並列計算機の場合には、送信側のメッセージパッシング型ライブラリ 1 4 1 が上記第 3 の `PUT` コマンドを発行するのに代えて、受信側のメッセージパッシング型ライブラリ 1 4 0 が `GET` コマンドを発行し、受信側の高速通信ライブラリ 1 3 5 が送信側のメモリ 1 2 5 から送信データを読み出す処理を実行する。

【0086】(8) メッセージパッシング型の通信におけるストリーム通信

図 4 に示されたデータ受信処理 7 0 7、送信処理 7 0 8 の詳細を図 7、8 に示した具体例を適宜参照して説明する。図 4 において、すでに述べたように送信側のアプリケーションプログラム 1 0 8 が、関数 `EMU_write` を呼び出した結果 (7 0 2)、送信処理 7 0 8 が起動されると、メッセージパッシング型ライブラリ 1 4 1 に含まれた関数 `MPI_Send` が呼び出される (7 8 1)。図 7 に示した例では、関数 `EMU_write` 呼び出し時 (7 0 2) の引数ではバッファ 8 0 3 のアドレス `Sbuff` と、5 0 キロバイト (KB) のバッファサイズを指定すると仮定する。関数 `MPI_Send` 呼び出し時 (7 8 1) の引数が指定するバッファアドレスとバッファサイズはこれらの値に等しくされる。以下では使用する関数の他の引数の説明は簡単化のために省略する。また、指定するバッファサイズの単位は KB であると仮定し、関数の引数を図示するときには、この単位 KB は簡単化のために図示しない。メッセージパッシング型ライブラリ 1 4 1 はこの関数呼び出し 7 8 1 に応答して、データの転送のために、既に述べたように高速通信機構 1 1 7 にヘッダの送信を指示し、その後データの



送信を指示する。

【0087】一方、図4において、すでに述べたように受信側のアプリケーションプログラム107が、関数EMU\_readを呼び出した結果(701)、受信処理707が起動されると、この受信処理707では図6に示す処理がなされる。なお、図7に示した例では、関数EMU\_readの呼び出し701の引数が指定するバッファアドレスはRbuffであり、バッファサイズは30KBであり、送信関数EMU\_writeの呼び出し702が指定したバッファサイズより小さいサイズを指定していると仮定する。

【0088】この受信処理707では、まず、受信側のTCP/IPエミュレーションライブラリ112内に設けられるデータ受信用の特定のバッファ(後に示すバッファ901)に受信済みでまだ受信側のアプリケーションプログラムに転送されていないデータが残っているかを判定する命令を発行する(処理202)。この命令は図7には示されていない。今仮定しているように最初に関数EMU\_readが実行されたときにはこの判定の結果は否定的となる。その後、送信データ検知命令MPI\_probe771が発行され、受信側のアプリケーションプログラムに宛てて送信されようとするデータがあるかを判別する(処理206)。この命令は、具体的には、このアプリケーションプログラム107に宛てて送信されるべきデータに関する、既に説明したヘッダが高速通信機構116により受信済みであるか否かを判定する命令である。もしこのヘッダが送信側のアプリケーションプログラム108からまだ送信されていない場合には、受信処理708は、処理を終了し受信側アプリケーションプログラム107に戻る(処理207)。受信側アプリケーションプログラム107は受信が失敗したときの処理を実行する。たとえば、受信が成功するまで関数701を繰り返して呼び出す。

【0089】ヘッダが送信側のアプリケーションプログラム108からすでに送信済みであると仮定すると、そのヘッダが指定する送信データが受信側のアプリケーションプログラムのバッファ812に入りきるか否かが判定される(処理208)。図7の例では送信データのサイズは、50KBであり、受信側のバッファ812のサイズは30KBであり、この判定の結果は否定的となる。このような場合に、受信側のTCP/IPエミュレーションライブラリ112が、この送信データを受信側のアプリケーションプログラム107のバッファ812に受信するMPI\_Recv命令を発行すると、受信側のメッセージパッシング型ライブラリ140は、通常はMPI仕様によりこの命令をエラーとして処理するかもしれない。もしくは送信データの内、受信側のアプリケーションプログラムのバッファ812に入りきらない分を捨ててしまう。

【0090】これを防ぐために、本実施の形態では受信

側アプリケーションプログラム107とリンクしている受信側TCP/IPエミュレーションライブラリ112内に、特別なバッファ901を用意し、受信側TCP/IPエミュレーションライブラリ112はここに送信データをこのバッファ901に一旦受信することを要求する。すなわち、このバッファ901のアドレスEbuffと全送信データのサイズ50KBとを指定する、関数MPI\_Recvを呼び出す(772)。

【0091】この関数呼び出し772に回答して、受信側的高速通信機構116と送信側的高速通信機構117は、すでに述べたようにしてデータを送受信し、受信側的高速通信機構116は、このデータを上記バッファ901に書き込む。受信側のTCP/IPエミュレーションライブラリ112はその後、この受信データの内、関数EMU\_Read呼び出し時の引数で指定されたサイズ30KBのデータを受信側のアプリケーションプログラム107のバッファ812にコピーするメモリコピー命令(MEMCPY(Rbuff, Ebuff, 30KB))773を発行する(処理209)。その後処理はアプリケーションプログラムに戻る(処理211)。こうして、TCP/IPエミュレーションライブラリ112内のバッファ901に受信されたデータの一部906が残っている状態で、アプリケーションプログラム間の送受信が完了する。

【0092】その後さらに送信側アプリケーションプログラム108がアドレスSbuff'のバッファ804のデータ80KBを送信するために関数EMU\_write(Sbuff', 80KB)(790)(図7

(B))を呼び出すと、同様にして送信処理708が実行され、この処理の中で関数MPI\_Send(Sbuff', 80)(791)が呼び出される。一方、受信側アプリケーションプログラム107もアドレスRbuff'のバッファ813に100KBのデータを受信するために関数EMU\_read(Rbuff', 100KB)(775)を呼び出すと、これに対しても同様に、受信707が実行される。

【0093】この受信処理707は、最初の判定処理202における判定では、受信側のTCP/IPエミュレーションライブラリ112内の受信用のバッファ901内に未転送のデータ906が残っていると判断される。その結果、メモリコピー命令memcpy(Rbuff', Ebuff+30KB, 20KB)(776)を発行して、このデータ906を受信側のアプリケーションプログラム107内のバッファ813の先頭領域907にコピーする(処理203)。このメモリコピーするデータの長さは、バッファ901に保持されているデータの長さの内、関数EMU\_readに対する上記第2の関数呼び出しが指定するバッファ長を超えない範囲に設定される。今の場合にはバッファ901に保持されているデータの長さが20KBであり、受信を要求された

データの長さが 1 0 0 K B より小さいので、このデータ 2 0 K B が全てコピーされる。

【0 0 9 4】その後受信側アプリケーションプログラム 1 0 7 が要求する 8 0 K B の残りのデータをさらに受信するために、処理 2 0 6 が実行される。この処理 2 0 6 ではすでに説明したように、送信データがあるか否かを調べる命令 `MP I _ P r o b e ( )` が発行される。具体的には、送信データに対するヘッダが受信済みであるか否かが判定される。今の場合に、送信側の関数呼び出し 7 9 1 がすでに実行済みであると仮定すると、8 0 K B の送信データがあることが判明する。その場合には、この送信データのサイズが受信側アプリケーションプログラム 1 0 2 が指定するバッファ 8 1 3 に入りきるか否かが判定される（処理 2 0 8）。今の場合には受信側のアプリケーションプログラムのバッファ 8 1 3 の残りの領域 9 0 8 のサイズは 8 0 K B なので、送信データはバッファ 8 1 3 のこの領域 9 0 8 に入りきる。

【0 0 9 5】この結果、関数 `MP I _ R e c v . ( R b u f f ' + 2 0 K B , 8 0 K B )` (7 7 8) が呼び出される。この関数呼び出しは、バッファ 8 1 3 の残りの領域 9 0 8 のアドレスと送信データのサイズ 8 0 K B を指定する。こうして、この送信データがバッファ 8 1 3 の領域 9 0 8 に直接受信される（処理 2 1 0）。その後、受信を要求されたデータが全て受信されたか否かが判定される（処理 2 1 2）。今の場合には判定の結果が肯定的であるので、受信処理 7 0 7 は終了し、処理はアプリケーションプログラムに戻る（処理 2 1 1）。

【0 0 9 6】以上の手順により、複数の関数 `EM U _ w r i t e` の呼び出し (7 0 2, 7 9 0) が指定する送信データをひと繋ぎのデータストリームとして複数の関数の呼び出し `EM U _ r e a d` (7 0 1, 7 7 5) により受信する、ストリーム通信を実現することができる。

【0 0 9 7】送信側のアプリケーションプログラムが指定する送信データのサイズが受信側のアプリケーションプログラムが指定するバッファのサイズよりも小さいときでも、以下のようにしてストリーム通信が簡単に実現される。たとえば、送信側のアプリケーションプログラムが 5 0 K B のデータの送信を繰り返し要求し、受信側のアプリケーションプログラムが 1 0 0 K B のデータの受信を要求する場合のストリーム通信を図 8 を参照して説明する。

【0 0 9 8】送信側のアプリケーションプログラムが呼び出す関数 `EM U _ w r i t e` (7 0 2) に対する送信処理 7 0 8 (図 4) の中で、関数 `MP I _ s e n d` (7 8 1) が呼び出される。この関数呼び出しでは、送信側アプリケーションプログラムのバッファ 8 0 3 のアドレス `S b u f f` とサイズ 5 0 K B を指定する。

【0 0 9 9】受信側のアプリケーションプログラムが呼び出す関数 `EM U _ r e a d` (7 0 1) に対する受信処理 7 0 7 (図 4) も、すでに述べたように図 6 に従い処

理される。今の仮定では処理 2 0 2 での判定は失敗する。処理 2 0 6 において、データ検査命令 7 7 1 が実行されたときに、送信データがあると判定されたと仮定する。今の場合には受信側のバッファ 8 1 2 のサイズは、送信側のバッファのサイズより大きいので、処理 2 0 8 での判定の結果は肯定的となる。その結果、処理 2 1 0 が実行される。この処理では、送信データを受信側アプリケーションプログラムが指定したバッファ 8 1 2 に直接受信するための関数 `MP I _ r e c v` (7 7 4) が呼び出される。この関数呼び出しは、受信側アプリケーションプログラムのバッファ 8 1 2 の先頭アドレス `R b u f f` と送信側のバッファ 8 0 3 のサイズ 5 0 K B を指定する。こうして、送信側のバッファ 8 0 3 内の全データが、受信側のバッファ 8 1 2 内の先頭の 5 0 K B の領域 9 0 7 に書き込まれる。次に処理 2 1 2 が実行される。今の場合、受信を要求されたデータのサイズは 1 0 0 K B であるのに対して、すでに受信されたデータのサイズは 5 0 K B である。したがって、要求されたデータの一部がまだ受信されていない。したがって、判定 2 1 2 の結果は否定的となり、残りのデータを受信するために処理 2 0 6 が再度実行される。

【0 1 0 0】もし、送信側のアプリケーションプログラムが次に関数 `EM U _ w r i t e` 7 9 0 を呼び出せば、それに対する送信処理 7 0 8 (図 4) の中で、関数 `MP I _ s e n d` (7 9 1) が同様に呼び出される。この関数呼び出しでも、送信側アプリケーションプログラムの次のバッファ 8 0 4 のアドレス `S b u f f '`  とサイズ 5 0 K B を指定する。

【0 1 0 1】上記処理 2 0 6 を繰り返したときに、すでに送信側のアプリケーションプログラムが上記次の関数 `EM U _ S e n d` 7 9 1 を呼び出していたならば、処理 2 0 6 での判定結果は肯定的となり、判定処理 2 0 8 に移る。今の場合には、受信側のバッファ 8 1 2 の残りの領域 9 1 0 のサイズは送信されようとするデータのサイズに等しいので、この判定の結果は肯定的となる。その結果、処理 2 1 0 が実行され、送信データを受信側のバッファ 8 1 2 の残りの領域 9 1 0 に直接書き込むための第 2 の受信関数 7 7 9 が呼び出される。この関数呼び出しでは、受信側アプリケーションプログラムのバッファ 8 1 2 の残りの領域 9 1 0 のアドレス `R b u f f + 5 0 K B` と送信データのサイズ 5 0 K B とを指定する。こうして、処理 2 1 2 において、要求された全てのデータの受信が完了したと判断されるので、受信処理 7 0 7 は完了する。なお、上記処理 2 0 6 が繰り返し実行された時点で送信データが存在しないときには、受信処理 7 0 7 は終了し、処理は受信側アプリケーションプログラムに戻る。また、上記処理 2 0 8 が繰り返された時点で、処理 2 0 8 での判定結果が否定的であるときには、処理 2 0 9 が実行される。この処理の内容は、すでに説明したものと同一である。以上のごとく、送信側のアプリケ

ーションプログラムが発行した送信命令EMU\_writeが指定するバッファのサイズと受信側のアプリケーションプログラムが発行した受信命令EMU\_readが指定するバッファのサイズが異なっている場合、また、送信側のアプリケーションプログラムが発行する送信命令EMU\_writeの数と受信側のアプリケーションプログラムが発行する受信命令EMU\_readの数が異なっている場合でもバッファストリーム通信が実現されることが分かる。

【0102】以上の説明から分かるように、本実施の形態によれば、並列計算機内部の様に、メッセージパッシング型通信方式の高速通信機構が提供されている計算機で動くアプリケーション同士が、TCP/IPを用いてデータ通信を行なう際に、高速通信機構の特徴を活かした高速通信が可能となる。また、それ以外の計算機上で動くアプリケーションとは、従来通りのTCP/IPによる通信を保証する。利用者は、既存のTCP/IPアプリケーションを一切変更する必要がない。

【0103】<発明の実施の形態1の変形例>本発明は、実施の形態1の内容に限定されるのではなく、以下に例示する変形例および他の変形例を含めいろいろの実施形態により実施できる。

【0104】(1) 広域ネットワークの通信規約としてTCP/IPを使用した、これに代えて他の通信規約を用いることもできる。そのときには、TCP/IP処理ルーチン、エミュレーションライブラリを変更する必要があるのは言うまでもない。

【0105】(2) 実施の形態1では、メッセージパッシング型ライブラリを使用した、これを使用しないことも可能である。このときには、エミュレーションライブラリは、直接高速通信ライブラリを呼び出すことになる。

【0106】(3) さらに、この通信ライブラリをなくすことも可能である。たとえば、これに代えて、専用の回路を使用することもできる。

【0107】(4) 実施の形態1では、内部計算機は全て広域通信網に接続されると想定した。しかし、一部の内部計算機が広域通信網に接続されている場合にも同様に本発明を適用できる。

【0108】(5) 実施の形態1では、広域通信網と内部高速通信網の両方を利用することを前提とした。しかし、本発明によるストリーム通信それ自体は、メッセージパッシング型の通信を実行可能な計算機間に適用できるものであり、したがって、このストリーム通信を実行するには、TCP/IP通信を使用しないでメッセージパッシング型の通信のみを使用するアプリケーションプログラム間の通信にも適用できる。この場合には複数種類の通信網を使用しなくてもよい。その際には、送信側のエミュレーションライブラリは実質的には使用しない変形例も可能である。

【0109】<発明の実施の形態2>TCP/IP処理ルーチンには従来からアプリケーションプログラムが使用可能な関数としてselect関数が設けられている。そもそもソケット記述子はファイル記述子の一種として定義されている。このファイル記述子が指定するオブジェクトから、データを取得することが可能であるかどうかを調べるためのシステムコールとして、select関数が用意されている。例えば、あるソケットから受け取り可能なデータが送信側から送られてきている

(あるいは送られようとしている) かどうか、select関数によって調べることができる。具体的にはあるソケットを割り当てられているアプリケーションプログラムが、送信システムコールwriteを発行したか否かが判定できる。select関数では、見張ろうとするファイル記述子をビット列で指定する。このビット列の各ビットはそれぞれ個別のファイル記述子に対応しており、ビットを1にすることでファイル記述子を指定する。複数のビットを1にすれば、一回のselect関数で複数のファイル記述子を同時に見張ることができる。select関数は、見張っているファイル記述子の何れかがデータ受け取り可能な状態になるまでブロックする。

【0110】発明の実施の形態1で用いるTCP/IPエミュレーションライブラリでは、並列計算機内部の通信時には従来のソケットライブラリを用いないため、selectシステムコールでは、内部通信用のソケットからデータ受信可能であるかどうかを調べることができない。そこで、本実施の形態では、実施の形態1のごとくTCP/IPエミュレーションライブラリを使用する計算機システムにおいても、アプリケーションプログラムが従来と同様にselect関数を利用可能にする。

【0111】本実施の形態では、内部ソケットテーブル410、411等に登録されているソケット記述子に対しては、内部通信専用のselectに相当する処理を行い、それ以外のファイル記述子に対しては、従来のselectシステムコールをそのまま用いる、という切り分けを行うselect関数EMU\_selectをTCP/IPエミュレーションライブラリ113内に設ける。

【0112】ただし、内部通信用のselectとselectシステムコールは同時に並行して実行しなくてはならない。2つのselectを逐次に行うのは、例えば内部通信用のselectがデータを待っている間は、外部との通信用のソケットや標準出力などがデータを受け取り可能な状態になった場合でも、それを検知することができないからである。

【0113】selectの同時実行を疑似的に実現する手段として、内部用selectとselectシステムコールをノンブロッキングで続けて発行することをスピループで繰り返す、という方法が考えられる。し

かしこの方法を用いると、データが受け取り可能になるまで計算機を占有してしまい、同一計算機上で走っている他のプロセスに処理が渡らなくなってしまう。

【0114】これに対して本実施の形態では、1ループ毎に処理を他のプロセスに譲渡する命令を挿入するという方法を採用。この方法により、スピングループによる計算機の占有を避けることができる。

【0115】図9は、図3で示した接続状態において、アプリケーションプログラム108および109が送信命令1002、1003を発行して、アプリケーションプログラム107にデータを送信し、アプリケーションプログラム107側でそれらのデータの到着を、select命令1001によって見張っている様子を表す。select命令1001で指定しているビット列はソケットSA0およびsa1に対応しているとする(1004、1005)。このうちsa1は内部ソケットテーブルに登録されているので、内部用selectで見張る(1008)。一方、SA0は内部ソケットテーブルに登録されていないので、selectシステムコールで見張る(1009)。TCP/IPエミュレーションライブラリで発行するselectシステムコールではsa1を見張る必要が無いので、アプリケーションプログラム107が発行したselect命令1001で指定されていたビット列に対し、sa1に対応するビットを0にしたビット列を指定する。処理1008と1009はノンブロッキングで発行し、交互に繰り返す(1010、1011)。ただし、繰り返しの途中で処理を一旦、他のプロセスに譲渡する。

【0116】図9に示したselect命令、内部用のselect関数をアプリケーションプログラムが使用可能にするためには、TCP/IPエミュレーションライブラリ112、113等にはエミュレーションselect関数EMU\_selectが設けられ、アプリケーションプログラムは、これを呼び出して使用する。アプリケーションプログラムがEMU\_select関数を呼び出す際には従来と同じく、それぞれ一つのソケットに対応するビットからなるビット列ap\_bitsを指定する。TCP/IPエミュレーションライブラリは、この関数呼び出しに応答して図10にともない処理を実行する。

【0117】まず、そのビット列ap\_bitsに対し(処理1101)、内部ソケットテーブルに登録されているソケット記述子に対応したビットを0にするためのマスクをかける(処理1102)。in\_maskは、内部ソケットテーブルに登録されている全てのソケット記述子に対応するビットが0、それ以外のビットが1であるようなビット列である。よって、ap\_bitsにin\_maskをかけて作成したビット列ex\_bitsは、ap\_bitsで指定されたファイル記述子のうち、内部通信用のソケット記述子を除いたビット列とな

る。その後、ex\_bitsを引数にしたselectシステムコールと、内部用select処理をノンブロッキングで一回ずつ実行し(処理1103、1104)、もし、この処理で調べたファイル記述子の何れかがデータ受け取り可能状態であった場合にはリターンする(処理1106)。そうでない場合は、一旦他のプロセスに処理を譲渡し(処理1107)、再びselect処理を繰り返す(処理1108)。

【0118】こうして、本実施の形態によれば、実施の形態1のように内部高速通信機構を併用する計算機システムにおいても、アプリケーションプログラムがselect関数を利用可能になる。

【0119】<発明の実施の形態3>上記スピングループによるselect関数の実現方法では、スピングループの途中で他のプロセスに処理を譲渡する処理1107を挿入することで計算機の占有を回避するが、同じ計算機上で処理されるプロセスの優先度が低いと、そのプロセスには処理が渡らない可能性がある。データの到着をスリープして待つブロッキングウェイトを用いればこれを避けることができるが、ブロッキングウェイトの内部通信用select処理とブロッキングウェイトのselectシステムコールを、1プロセス・1スレッド上で同時に実行することはできない。

【0120】これに対して本実施の形態では、まず2つのスレッドを生成し、一方のスレッド上では内部用selectを、もう一方ではselectシステムコールを実行するという方法を採用。この方法によれば、内部用selectとselectシステムコールがそれぞれのスレッド上で独立に動作することができるため、同時にブロックしてデータを見張ることができる。

【0121】図11は、図9で示したのと同じ通信状態を表している。ただし、内部用selectとselectシステムコールの同時実行の実現方法が異なる。図11では、これら2つのselect処理は、別々のスレッドの上で実行する(1203、1204)。処理1009と同様に、スレッド1203上のselectシステムコールでは、sa1に対応するビットを0にしたビット列を指定する。

【0122】図12を参照するに、本実施の形態において、アプリケーションプログラムがEMU\_select関数を発行する際に指定したビット列ap\_bitsに対し、内部ソケットテーブルに登録されているソケット記述子に対応したビットを0にするためのマスクをかけてex\_bitsを作成する処理までは図10の処理(1101、1102)と同じである(処理1301、1302)。その後、まずノンブロッキングでselectシステムコール(処理1303)と内部用select処理(処理1304)を1回ずつ行う。この処理で調べたファイル記述子の何れかがデータ受け取り可能状態であった場合にはリターンする。そうでない場合に

は、スレッドを2つ生成し(処理1307)、それぞれの上でselectシステムコール(処理1308)と内部用select(処理1309)を実行する。これらの処理は、それぞれデータ受け取り状態になるまでブロックする。両処理のうち先にブロックが解けた方は、もう一方のスレッドをキャンセルして(処理1310、1311)リターンする(処理1314、1315)。このキャンセル処理では、スレッドを強制的に終了させるのではなく、そのスレッドがもう不要であるという印を付ける。スレッドはブロックが解けた時にこの印が付けられているかどうかを調べ(処理1316、1317)、もし付けられていれば、キャンセルされていたことになるのでそのまま消滅する(処理1318、1319)。

【0123】上記select処理において、スレッド分割の前に処理1303および処理1304のノンブロッキングselect処理を一回ずつ行うのは、次の理由による。もし、処理1301が発行される以前に内部用ソケットとそれ以外のファイル記述子が共にデータ受け取り可能な状態になっている場合、select関数はその両方を検知できなければならない。しかし、いきなりスレッドを分割して、内部ソケットとそれ以外のファイル記述子を別々に見張り始めると、両スレッドのうち若干早く検知した方がもう一方のスレッドをキャンセルしてしまうため、片方のスレッドの状態しか検知することができない。これに対して処理1303および処理1304を実行することで、処理1301が発行される以前の内部ソケットとそれ以外の記述子の状態を両方とも確実に調べることができる。

【0124】本方式では、データの到着をスリープして待つため、実施の形態2のスピンループで待つ方法に比べると、データ検出のタイミングが遅れるが、優先度の低いプロセスに対しても、処理の妨げとなることを回避できる。

【0125】

【発明の効果】本発明によれば、ストリーム通信をメッセージパッシング型の通信でもって実現できる。

【0126】本発明の他の態様によれば、第1の通信網とそれより高速の第2の通信網に接続された計算機上で動作するアプリケーションプログラムが、第1の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間で第1の通信規約に基づいて通信することができ、さらに、第2の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間でその第2の通信網を使用した高速の通信を行う

ことができる。

【0127】さらに具体的には、上記第1の通信規約はTCP/IP通信規約を使用できる。また、上記第2の通信網を使用した通信を、メッセージパッシング型の通信とすることができる。

【図面の簡単な説明】

【図1】本発明の実施例の全体構成図。

【図2】ソケット接続のフローチャート。

【図3】ソケット接続を説明するための図。

【図4】内部・外部通信切り分け方法のフローチャート。

【図5】内部・外部通信切り分け方法を説明するための図。

【図6】ストリーム通信のフローチャート。

【図7】送受信動作に使用される命令列の一例を示す図。

【図8】送受信動作に使用される命令列の他の例を示す図。

【図9】select機能をスピンループで実現する方法を説明するための図。

【図10】select機能をスピンループで実現する方法のフローチャート。

【図11】select機能をスレッドを用いて実現する方法を説明するための図。

【図12】select機能をスレッドを用いて実現する方法のフローチャート。

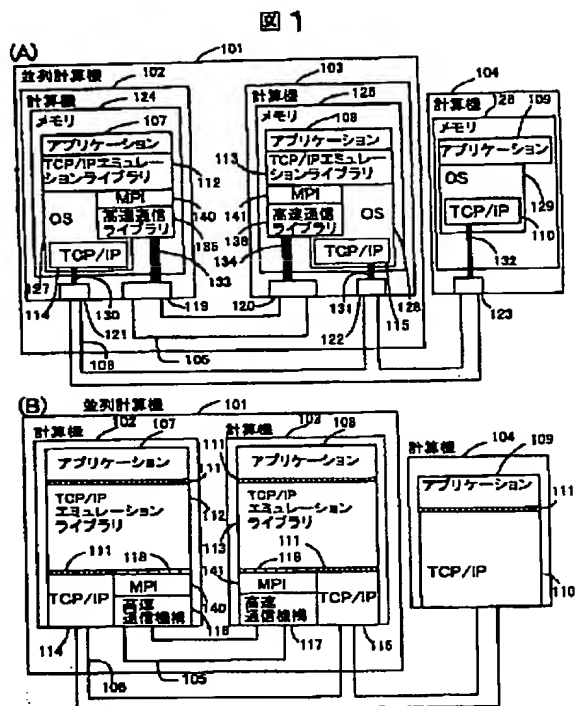
【図13】従来のTCP/IP通信規約の階層図。

【図14】従来のストリーム通信を説明するための図。

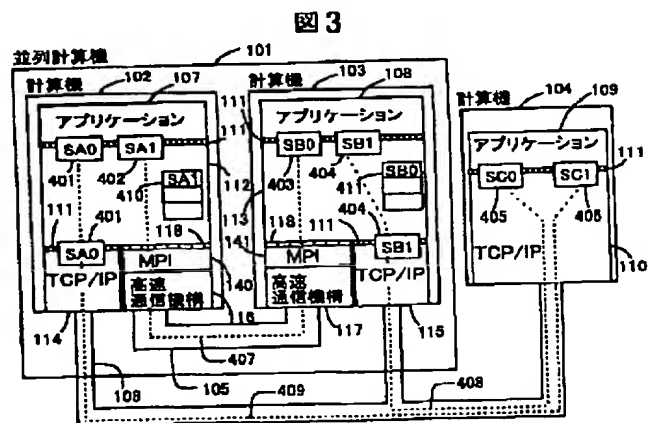
【符号の説明】

105... 並列計算機内部通信網, 106... グローバル通信網, 111... ソケットアプリケーションプログラムインタフェース, 118... MPI仕様のインタフェース, 119, 120, 121, 122, 123... ネットワークインタフェースハードウェア, 140, 141... メッセージパッシング型ライブラリ, 401, 402, 403, 404, 405, 406... ソケット, 407, 408, 409... ソケットのコネクション, 410, 411... 内部ソケットテーブル, 601... 並列計算機内部通信時におけるデータ経路, 602... 外部計算機との通信時におけるデータ経路, 803, 804, 812, 813... アプリケーションプログラムのバッファ, 809... ストリーム, 901... TCP/IPエミュレーションライブラリ内のバッファ。

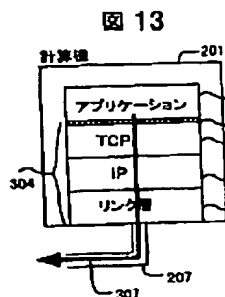
【図 1】



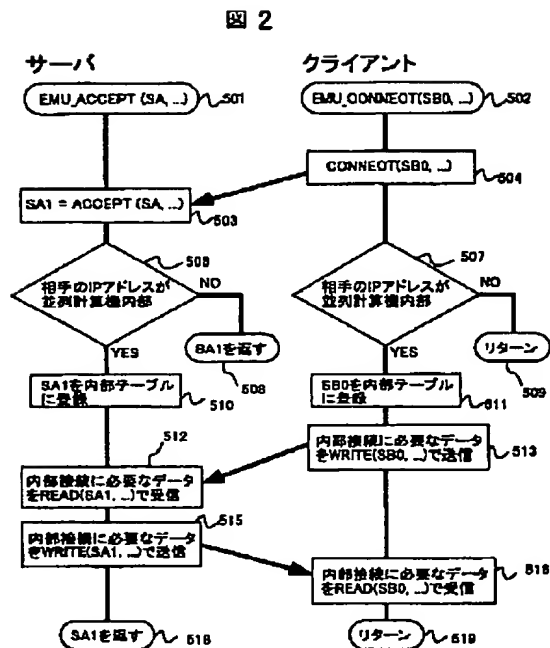
【図 3】



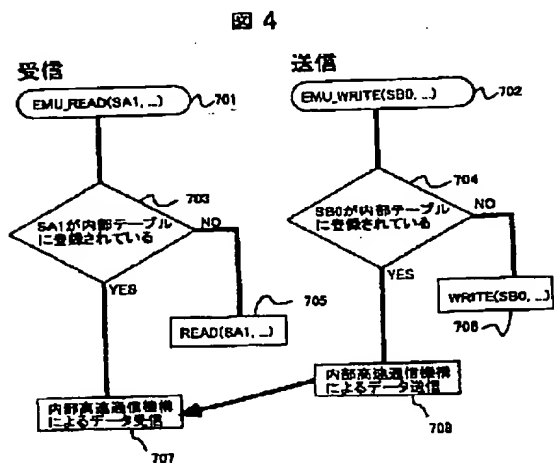
【図 13】



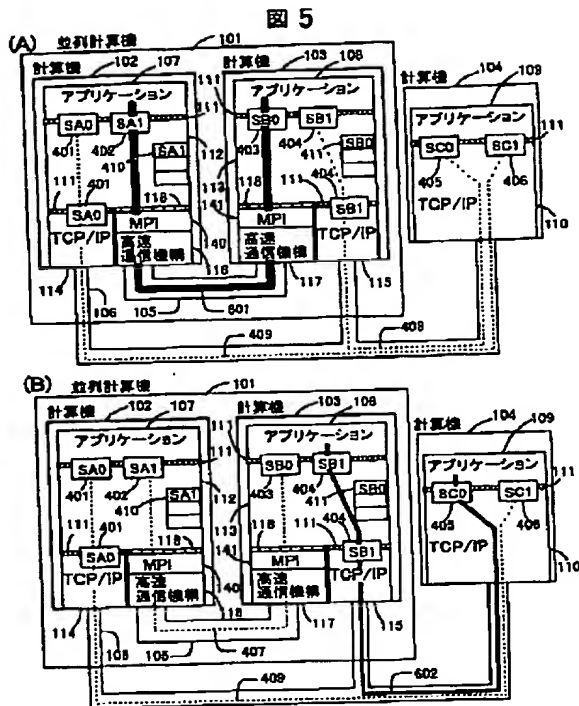
【図 2】



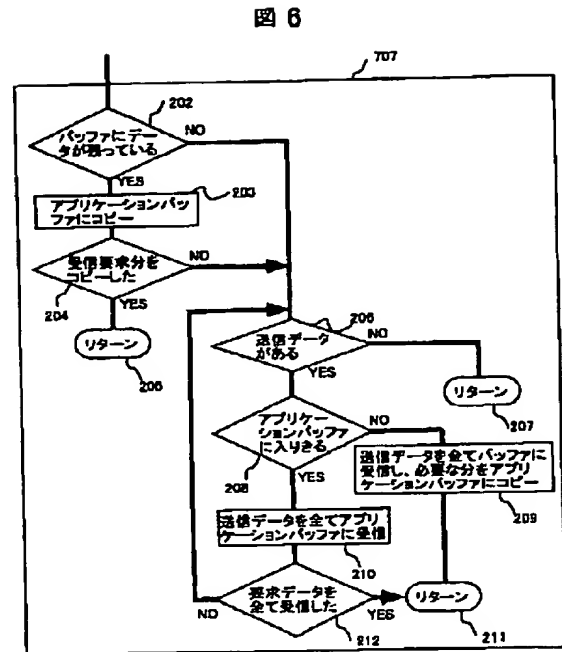
【図 4】



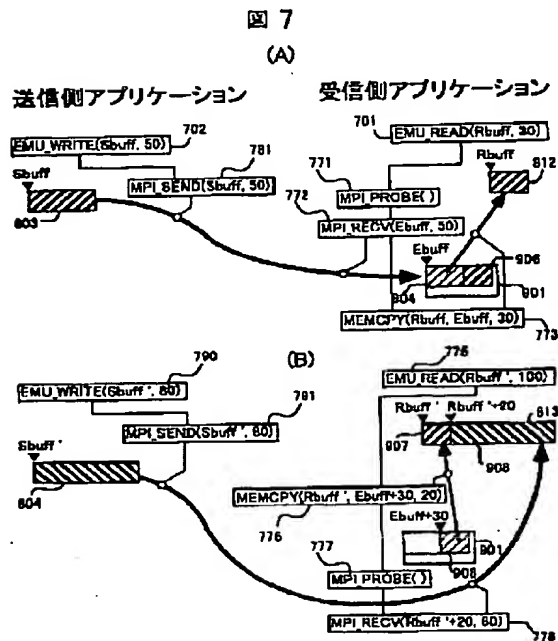
【図 5】



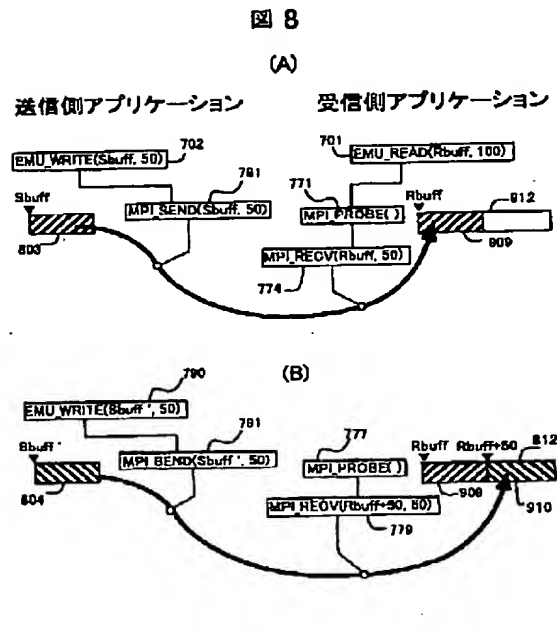
【図 6】



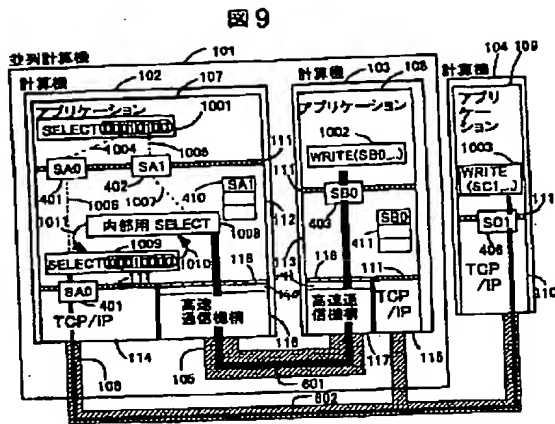
【図 7】



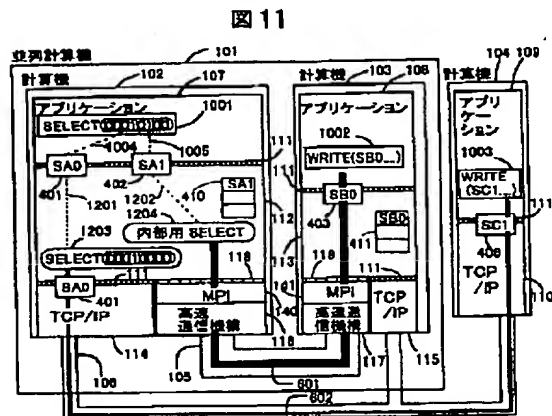
【図 8】



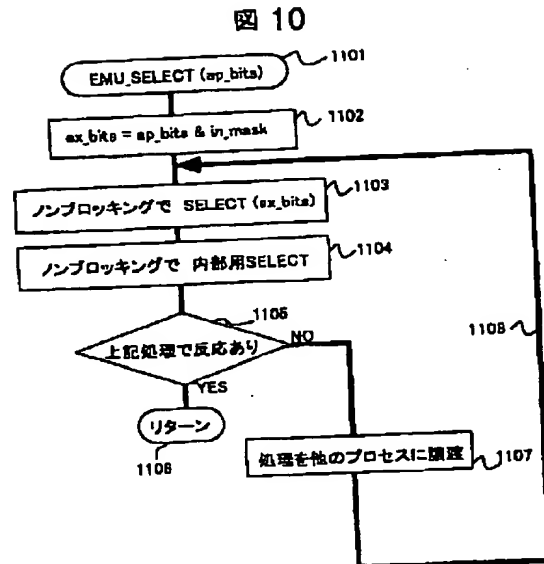
【図9】



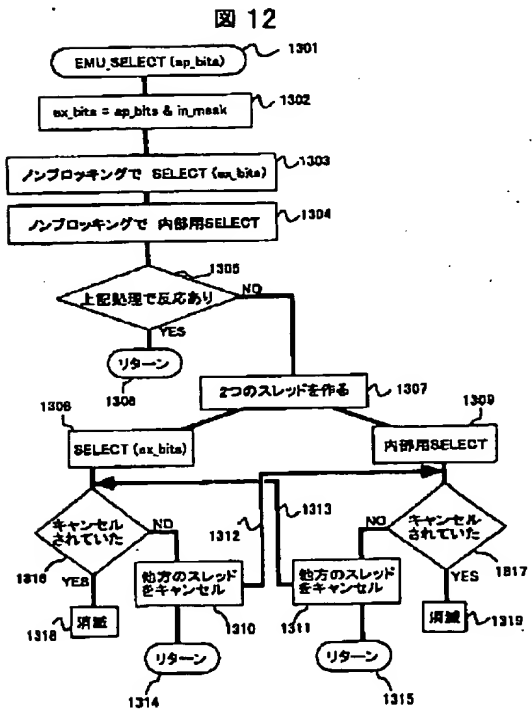
【図11】



【図10】



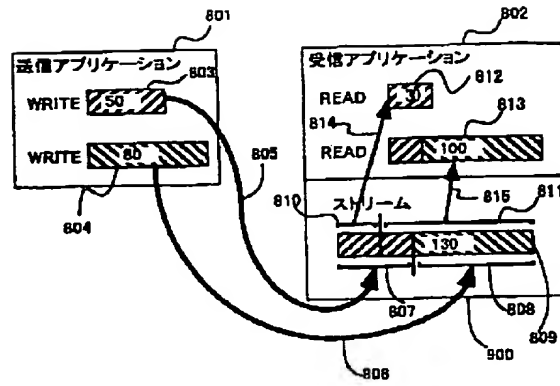
【図12】





【図 1 4】

図 14



**THIS PAGE BLANK (USPTO)**

**THIS PAGE BLANK (USPTO)**